Universitat de Lleida

TREBALL FINAL DE GRAU

ESCOLA
POLITÈCNICA SUPERIOR
UNIVERSITAT DE LLEIDA
INSPIRING THE FUTURE

Estudiant: Nil Torrent Bureu

Titulació:

Títol de Treball Final de Grau: **Disseny eficient de xarxes neuronals per a problemes de grafs**

Director/a: Josep Argelich Romà i Jordi Planes Cid

Presentació
Mes: Octubre

Any: 2020

Disseny eficient de xarxes neuronals per a problemes de grafs

Treball de final de grau - Grau en Enginyeria Informàtica

Nil Torrent

Índex

Índex de figures

Capítol 1

Resum/abstract

El problema que s'estudia en aquest treball per posar a prova el model de xarxa neuronal per a grafs es basa fonamentalment en el paper "An Argumentation Approach for Agreement Analysis in Reddit Debates"[1], realitzat pel grup de recerca GREiA, el qual és un de diversos papers d'una línia de recerca on s'analitzen converses de xarxes socials, en aquest cas centrant-se en la xarxa social Reddit.

Molt resumidament, en aquest paper s'analitzen converses reals de Reddit per comprovar quina és la posició guanyadora i el nivell de discrepància entre usuaris. Es representa cada conversa utilitzant diversos tipus de grafs, l'últim dels quals es modela com un *argumentation framework*. Llavors, es calculen els comentaris (nodes) acceptats utilitzant les *ideal semantics* del *abstract argumentation framework*. Finalment, es comparen diferents característiques dels comentaris acceptats i es mesura la polarització normalitzada en el conjunt de comentaris acceptats.

En aquest treball, també es parteix de diverses converses de Reddit modelades com a graf, i es vol acabar obtenint la polarització normalitzada de cada conversa, però utilitzant una xarxa neuronal per a grafs (GNN). Es vol provar de determinar directament la polarització normalitzada de cada conversa a partir d'una de les modelitzacions inicials de la conversa com a graf. És a dir, es prescindiria dels càlculs entre les primeres modelitzacions d'una conversa com a graf i el resultat final, en el que destaca la part més costosa: utilitzar ïdeal semantics"del "abstract argumentation framework".

En aquest projecte s'ha dissenyat i parametritzat un model de xarxa neuronal, la qual donada una conversa de Reddit en forma de graf, ha de determinar la polarització normalitzada de la conversa.

Per a dur-ho a terme s'han de recollir una gran quantitat de converses de reddit, cercar diversos models de GNNs existents i escollir i optimitzar el model més adequat que es pugui adaptar a les premises del problema.

Capítol 2

Introducció i objectius

2.1 Introducció

Un dels camps més actius de recerca en Enginyeria Informàtica és l'aprenentatge automàtic (Machine Learning), i en particular, les xarxes neuronals. Aquestes han resultat útils per a dur a terme activitats intel·ligents relacionades amb imatges o vídeos. Activitats que són un desafiament actualment són aquelles relacionades amb elements discrets, tals com grafs. Exemples d'aplicacions poden ser els grafs generats a partir de cridades a funció d'una aplicació, relació entre arguments de converses en una xarxa social, o similitud entre vins produïts en una mateixa regió.

L'objectiu d'aquest treball és dur a terme en el disseny i parametrització de xarxes neuronals que tinguin com a entrada un graf, així com una codificació eficient del problema a l'entrada de la xarxa neuronal.

S'ha decidit treballar amb les converses directament en forma de graf, la seva representació més natural, i el problema consisteix determinar quina és la polarització normalitzada de cada conversa (és a dir, de cada graf).

2.2 Motivació: descripció del problema

Aquest treball pretén resoldre el problema de trobar la polarització normalitzada d'una conversació de Reddit utilitzant xarxes neuronals per a grafs.

En el paper *An Argumentation Approach for Agreement Analysis in Reddit Debates*[1] es descriu formalment el problema a resoldre, el que aquest treball aporta a la línia de recerca és la utilització de les xarxes neuronals per a grafs per a resoldre aquest mateix problema.

Les motivacions d'aquest treball són les següents:

- Aprendre sobre xarxes neuronals (diferents arquitectures i maneres d'optimitzar-les al màxim per aconseguir els millors resultats possibles)
- Utilitzar xarxes neuronals que utilitzin com a input directament un graf, és a dir, (GNN). En aquest tipus de xarxes neuronals hi ha hagut i està havent-hi molta investigació i nous avenços recentment, per tant està a l'ordre del dia.
- Aprendre a definir el model de dades i a fer el preprocessat de les dades a introduir a la xarxa, ja que s'utilitzarà un dataset propi.
- Definir un model de xarxa neuronal que accepti un graf arbitrari com a entrada.

Un dels objectius més rellevants d'aquest treball és permetre trobar la polarització d'un graf (conversa) arbitrari en temps polinòmic. L'algorisme distribuït (que la xarxa neuronal substitueix en aquest treball) permet trobar la solució correcta en temps polinòmic només si el graf no té cicles o és bipartit. Per tant, com que una xarxa neuronal entrenada determina la sortida amb un cost lineal, si s'aconsegueix una xarxa amb un alt nivell de precisió i un model que accepti com a entrada un graf arbitrari, podríem donar la solució al problema amb un cost lineal per un graf arbitrari i no només per a grafs sense cicles o bipartits. S'explica aquest tema més en detall en la secció 3.

Capítol 3

Descripció ampliada del problema original

3.1 Model de dades *orientat als comentaris* vs *centrat en l'autor*

El model de dades orientat als comentaris consisteix en un graf on cada node representa un comentari i les arestes representen la contestació d'un comentari a un altre. El model centrat en l'autor consisteix en un graf on cada node representa un usuari (que pot haver fet un o més comentaris) i una aresta representa que un usuari ha respost al comentari d'un altre usuari.

Des del punt de vista computacional, la principal diferència entre el model orientat als comentaris i el model centrat en l'autor rau en l'estructura del gràfic subjacent. Per al cas del model orientat als comentaris, sempre és un gràfic acíclic dirigit; no obstant això, per al cas del model centrat en l'autor, el gràfic normalment conté alguns cicles.

Una conversa centrada en l'autor sí que pot tenir cicles, ja que dos o més usuaris es poden contestar entre ells. En canvi, en el model orientat als comentaris, com cada comentari és una unitat diferent, no poden haver-hi cicles.

En aquest treball s'ha decidit utilitzar el model de dades orientat als comentaris, tot i que el model de xarxa neuronal haurà de permetre acceptar també el model centrat en l'autor.

3.2 Sistema de recuperació de debats (Debate retrieval system)

3.2.1 Debat original

Definim un debat de reddit Γ és un conjunt no buit de comentaris, els quals s'originen a partir del comentari arrel r que conté l'enllaç a la notícia que es discuteix. Un comentari $c \in \Gamma$ és una tupla $t = (m, a, s)$, on m és el text del comentari, a és l'autor del comentari, i $s \in \mathbb{Z}$ és la puntuació (score) que els usuaris han donat al comentari, la qual pot ser tan negativa com positiva.

3.2.2 Debate tree

Definim un debat tree $DebtT$ per a un debat Γ com una tupla $\langle C, r, E, P\rangle$ tal que

- a cada comentari de Γ li correspon un node a C
- r és el comentari arrel, el qual també està inclòs com un node a C
- $\forall(c_1, c_2) \in C$ on el comentari c_1 respòn al c_2, hi ha una aresta dirigida (c_1, c_2) (de c_1 a c_2) al conjunt E
- P és una funció d'etiquetatge $P : E \to [-2, 2]$, on s'assigna un valor entre -2 i 2 que denota el sentiment de la resposta, la qual va entre un sentiment altament negatiu (-2) a un altament positiu (+2).

3.2.3 Polarized debate tree (PDebtT)

Definim un Polarized debate tree $PDebtT$ per a un Debate tree $DebtT$ $\langle C, r, E, P\rangle$ respecte a un llindar α com una tupla $\langle C_\alpha, r, E_\alpha, P\rangle$ tal que:

- El conjunt de comentaris filtrats C_α i el conjunt d'arestes filtrats E_α ara estan filtrats pel llindar α. Tant C_α com E_α seran subarbres de C i E. El filtratge es defineix de la forma següent:
  - C_α només contindrà els comentaris $c \in C$ tal que, en el camí entre el comentari c fins al comentari arrel r, cada aresta (c_i, c_j) en el camí satisfà que $|P((c_i, c_j))| > \alpha$. És a dir, si comencem des del node arrel, una vegada trobem un node fill el qual tingui una aresta cap a ell amb un valor assignat menor a α, tot el possible subarbre que hi hagi sota el node fill s'eliminaria.
  - E_α només conté les arestes de E que connectin comentaris del conjunt filtrat C_α.

3.2.4 Weighted two-sided debate tree o Weighted Bipartite Debate Graph (WBDebG)

No s'utilitza en aquest treball per a estalviar conversions innecessàries en les dades que s'entren a la xarxa neuronal.

3.2.5 Paràmetre *alpha*

Cal tenir en compte el paràmetre α (el qual s'explica en l'apartat 3.2.3) modifica la polarització normalitzada de la conversa si el valor és prou alt per a ometre gran part dels comentaris de la conversa. Durant el treball, el valor α ha de ser prou baix per tal que la polarització normalitzada sigui la més pròxima possible a la conversa original sense filtrar.

3.3 Sistema de raonament (Reasoning system)

- Conversió a VAF

Es mapeja el nostre problema a una *valued abstract argumentation framework*(VAF)[2].

- Complexitat dels algorismes - Ideal semantics

La complexitat dels algorismes a substituir, els quals utilitzen *ideal semantics*[3] per a la VAF, tenen complexitat polinòmica o no segons les característiques del graf d'entrada:

Cal distingir dos tipus de grafs:

1) Grafs no bipartits amb cicles de llargada parell

En aquest cas per trobar la solució cal utilitzar un algorisme basat en ASP (*Answer Set Programming*) que s'explica a [4]. Tot i això, en aquest cas, no existeix cap algorisme que ho pugui resoldre en temps polinòmic. Per tant, utilitzant una xarxa neuronal que permeti entrar grafs d'aquest tipus podríem passar a resoldre el problema amb temps polinòmic. Com ens interessa resoldre aquest cas en temps polinòmic i és teòricament possible resoldre-ho utilitzant xarxes neuronals, el model de xarxa neuronal escollit en aquest treball permetrà aquest tipus de grafs com a entrada.

2) La resta (grafs sense cicles, amb cicles de llargada imparell, o bipartits amb cicles de llargada parell)

En aquest cas es pot utilitzar l'algorisme distribuït que s'explica a [5]. Aquest algorisme sí que és capaç de resoldre el problema en temps polinòmic. Com les converses de Reddit orientades a comentaris no tenen cicles, aquest serà el cas que s'estudiarà en profunditat.

3.4 Comparació dels sistemes de recuperació de debat i de raonament originals amb els quals cal implementar

En aquest apartat es compara el Sistema de recuperació de debats (apartat 3.2) i el Sistema de raonament (apartat 3.3) del paper original, amb què es vol aconseguir en aquest treball.

En la figura 3.1 es pot observar l'esquema del Sistema de recuperació de debats i el Sistema de raonament que s'utilitzen en el paper original. En aquest cas, el Sistema de recuperació de debats parteix de la conversa de Reddit original (Debat de Reddit) i com a sortida deixa el graf de tipus *Weighted Bipartite Debate Graph*.

En el Sistema de raonament s'inserta el *Weighted Bipartite Debate Graph*, es mapeja a un VAF, al qual s'aplica ideal semantics i té com a sortida un conjunt de comentaris acceptats. Tenint el conjunt de comentaris acceptats, s'obté la polarització normalitzada de la conversa amb un càlcul simple.

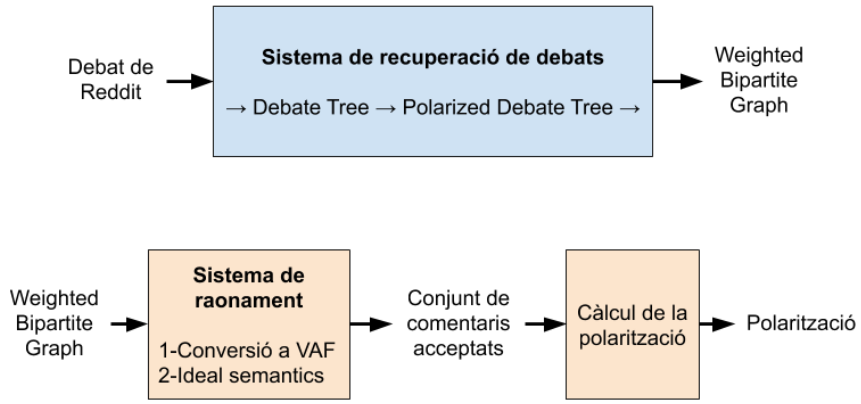

Figura 3.1: Esquema del Sistema de recuperació de debats i el Sistema de raonament en el paper original

En la figura 3.2 es pot observar l'esquema del Sistema de recuperació de debats i el Sistema de raonament que cal desenvolupar en aquest treball, el qual es pot comparar fàcilment amb la figura 3.1 per veure quines parts són les que arriba a substituir la xarxa neuronal. En aquest cas, Sistema de recuperació de debats parteix també de la conversa de Reddit original (Debat de Reddit) però com a sortida deixa el graf de tipus *Polarized debate tree*, s'omet la conversió final a *Weighted Bipartite Debate Graph*.

En el Sistema de raonament s'inserta el *Polarized debate tree*, i la xarxa neuronal substitueix tot el procés fins arribar a la polarització normalitzada.

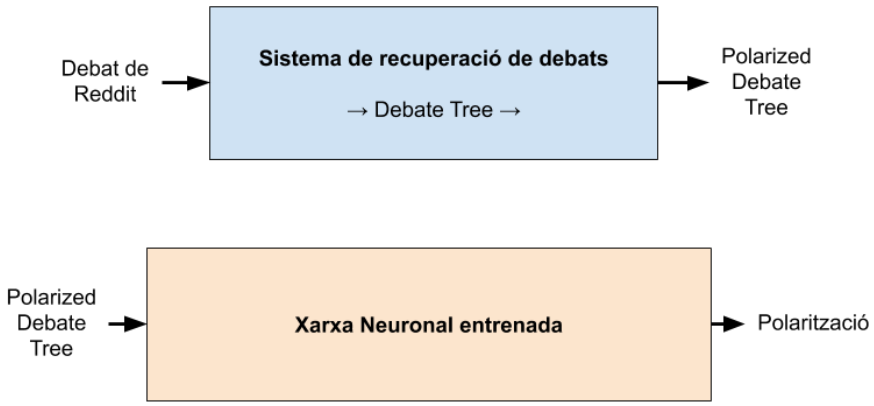

Figura 3.2: Esquema del Sistema de recuperació de debats i el Sistema de raonament a desenvolupar en el treball

Capítol 4

Reddit i el model de dades

4.1 Xarxa social reddit

Reddit és una de les xarxes socials més utilitzades actualment, amb més de 430 MILIONS d'usuaris actius cada mes (dades del 2020 extretes de la pàgina redditinc.com). S'ha triat analitzar converses de Reddit perquè ens interessa analitzar discussions amb un gran nombre de comentaris, i comentaris amb un límit de caràcters molt alt. Aquestes són concretament les raons:

- Límit de 40000 caràcters per cada comentari.
- Un comentari només pot contestar a un altre comentari (graf en forma d'arbre, grau de sortida de cada vèrtex és sempre 1 excepte el node arrel, el qual és 0), però un comentari pot ser respost per un nombre indeterminat de comentaris (grau d'entrada entre major o igual a 0).
- Els comentaris només poden contenir text (en cas d'acceptar imatges, animades o no, seria complicat fer el sentiment analysis). Tot i això, si es permet inserir enllaços.

I més concretament, s'extreuran converses únicament del subreddit "r/worldnews", on tots els posts que hi hagin han de complir les normes següents:

- El comentari que inicia el post (el vèrtex arrel), ha de consistir en una notícia de qualsevol tema. El comentari arrel sempre consisteix en el títol de la notícia I l'enllaç a la notícia. A més a més, la notícia ha de complir certs requisits com no utilitzar un títol enganyós, no ser editorials o articles d'opinió, no es pot incloure contingut ofensiu o discriminatori contra un grup determinat, I la notícia no pot ser de fa més d'una setmana.
- Les normes del subreddit també indiquen que els comentaris no poden consistir en atacs personals entre usuaris de reddit, sempre es discuteix a partir de la notícia

del comentari arrel.

El sentiment analysis del vèrtex arrel, la notícia, no es tindrà en compte.

4.2 Exemple d'una conversa de Reddit

S'ha triat una conversa de les que formen el dataset com a exemple per a mostrar les diferents modelitzacions per les quals passa cada conversa fins a arribar a l'estructura necessària per ser processada per la xarxa neuronal.

4.2.1 Modelitzacions de la conversa

- Debat original

Es pot trobar la conversa original prement sobre aquest enllaç.

- Debate tree (DebT)

A la figura 4.1 es mostra la conversa en forma d'arbre amb tots els comentaris que formen la conversa original.

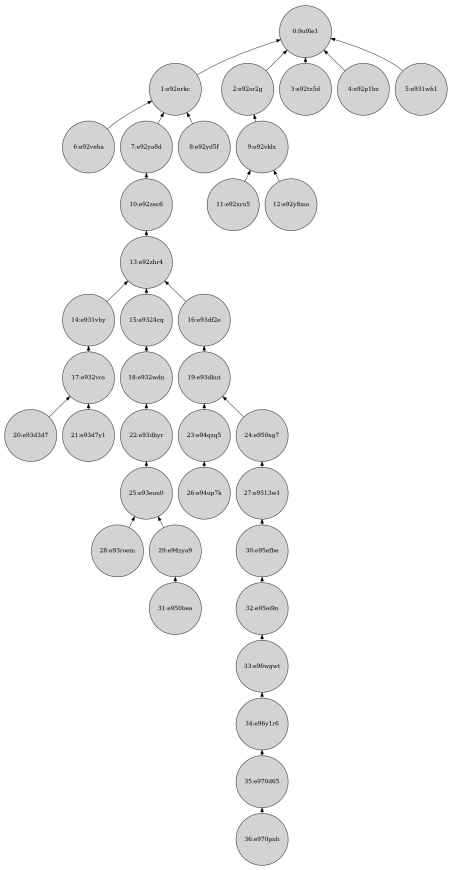

Figura 4.1: Conversa en la forma DebT

- Polarized debate tree (PDebtT)

En les figures 4.2, 4.3, 4.4 i 4.5 es mostren les converses en forma polaritzada i filtrada (PDebtT) amb un llindar de $\alpha = 0.5$, $\alpha = 0.05$, $\alpha = 0.7$ i $\alpha = 1.4$, respectivament.

Com es pot observar, en els 4 casos, fins i tot el cas on $\alpha = 0.05$, una gran part dels comentaris es filtren per no tenir prou rellevància en la conversa, però en el cas on $\alpha = 0.05$ la conversa queda reduïda únicament al comentari arrel i un comentari que el respon, el qual donarà una polarització normalitzada situada en un dels extrems (-1 o 1) però molt possiblement no serà realista respecte a la conversació original sense filtrar.

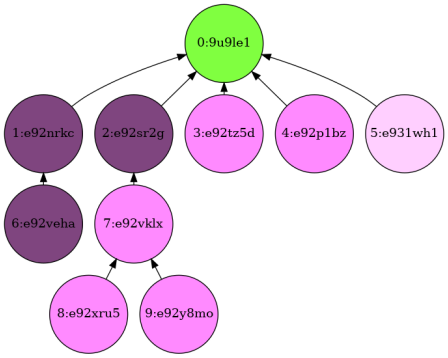

Figura 4.2: Conversa en la forma PDebT amb $\alpha = 0.5$

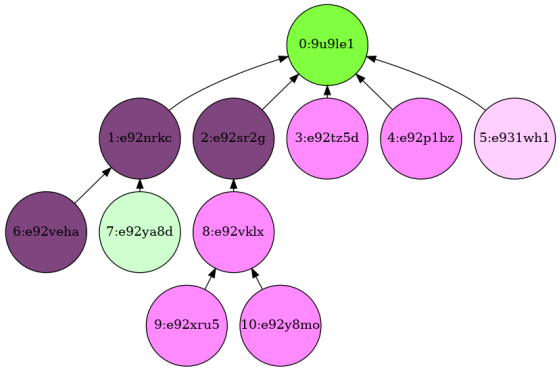

Figura 4.3: Conversa en la forma PDebT amb $\alpha = 0.05$

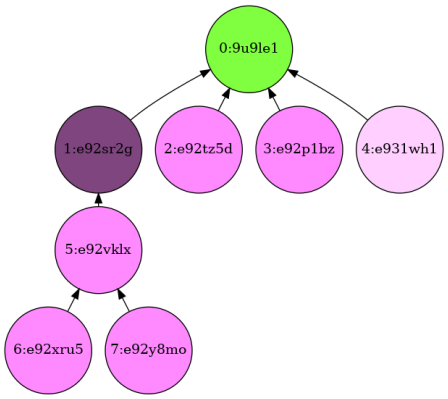

Figura 4.4: Conversa en la forma PDebT amb $\alpha = 0.7$

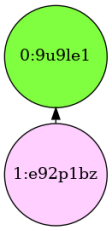

Figura 4.5: Conversa en la forma PDebT amb $\alpha = 1.4$

Capítol 5

Teoria bàsica de grafs

5.1 Definició de graf

Un graf G està format d'un parell (V, E) amb un conjunt finit de vèrtexs V i arestes E. Als vèrtexs també se'ls sol anomenar nodes.

Anomenem "mida d'un graf"al nombre d'arestes d'un graf, i es representa com a $|E(G)|$. Anomenem "ordre d'un graf"al nombre de vèrtexs d'un graf, i es representa com a $|V(G)|$.

5.2 Grafs dirigits i no dirigits

En grafs no dirigits, els elements que uneixen els vèrtexs (és a dir, les arestes) no tenen cap direcció definida. En aquest cas definiríem els conjunt d'arestes E com $E \subseteq \{\{u, v\} \in V | u \neq v\}$.

En canvi, en els grafs dirigits, les arestes poden tenir dues direccions definides. En aquest cas definiríem els conjunts d'arestes E com $E \subseteq \{(u, v) \in V \times V | u \neq v\}$.

En un graf dirigit, anomenem "grau de sortida"al nombre d'arestes sortints d'un vèrtex, i "grau d'entrada"al nombre d'arestes entrants en un vèrtex.

5.3 Grafs etiquetats i no etiquetats

Definim un graf etiquetat com una tripleta (V, E, l) amb una funcio l tal que $l : V(G) \cup E(G) \rightarrow \Sigma$, essent Σ un alfabet finit qualsevol. Llavors, *l(x)* és una etiqueta de x per $\forall x \in V(G) \cup E(G)$. Per tant, podem posar una etiqueta a qualsevol vèrtex o aresta del nostre graf.

Un graf no etiquetat es denota igual que amb la definició bàsica de graf descrita en la *secció 2.1.1.*

5.4 Arbres dirigits

Un arbre (dirigit) és un graf sense cicles. Sent p un vèrtex en un arbre dirigit, llavors anomenem als veïns de s "fills amb pare s". Si s no és fill d'un altre vèrtex, llavors seria l'**arrel** de l'arbre.

5.5 Representació en forma de graf de les converses de Reddit

Un post de Reddit és el que anomenem çonversació". Un post consisteix en un commentari arrel, al qual la resta d'usuaris responen. Al subreddit "worldnews"el comentari arrel sempre consisteix en una notícia de qualsevol medi digital. Llavors la resta d'usuaris poden respondre al comentari original, i llavors respondre's entre ells.

Per tant, la conversació (post) mínim consisteix en un graf amb un sol vèrtex, el vèrtex arrel, que és el comentari que ha comentat la notícia. En el nostre dataset les converssacions mínimes s'obvien, ja que volem almenys un graf amb 2 vèrtexs (el vèrtex arrel i una resposta al vèrtex arrel).

El grau de sortida d'un vèrtex sempre és 1 (ja que com tractem un comentari com a un vèrtex, un comentari només pot respondre a un altre sol comentari), excepte el vèrtex arrel, el qual té un grau de sortida igual a 0. El grau d'entrada de qualsevol vèrtex pot ser igual o més gran que 0, no hi ha un límit establert.

Per aquest motiu, podem modelitzar les conversacions de reddit com a arbres dirigits, el que anomenem *debate tree* tal com s'ha explicat en l'apartat 3.2.2.

Per mostrar totes les característiques d'una conversa de forma visual, s'ha utilitzat la mateixa conversa d'exemple de l'apartat 4.2, concretament en la forma PDebT amb $\alpha = 0.5$. Es pot veure en la imatge 5.1.

Aquests són els 5 aspectes del graf de la imatge 5.1:

- Identificador de node: en la primera línia del text de cada vèrtex, va del 0 (el node arrel) al 9.
- Matriu d'adjacència: es formaria amb arestes dirigides que es veuen
- Sentiment: en la tercera línia del text de cada vèrtex, en aquest cas és el sentiment no normalitzat, i en tots els vèrtexs és negatiu, curiosament.
- Score: en la segona línia del text de cada vèrtex.
- Polarització normalitzada: es troba en la descripció de la imatge, el valor és -0.09.

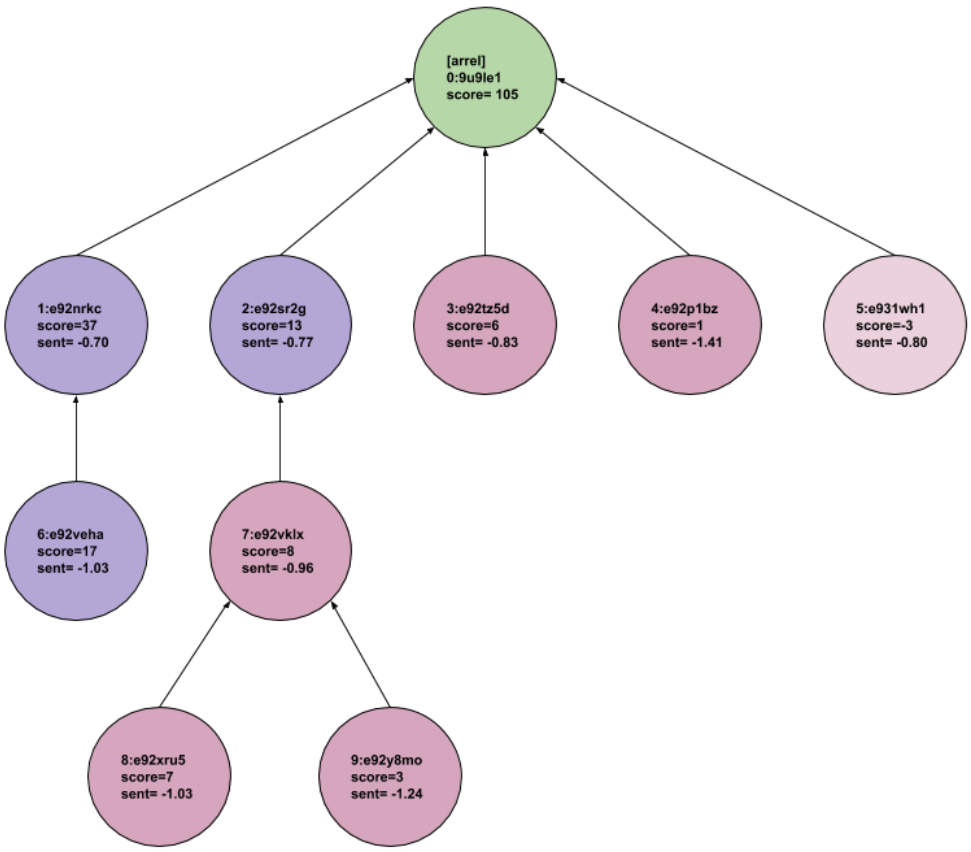

Figura 5.1: Conversa "9u9le1"en la forma PDebT amb $\alpha = 0.5$, amb les característiques de cada vèrtex (sentiment i score). La polarització normalitzada d'aquesta conversa és de -0.09

5.6 Isomorfisme de grafs

Un isomorfisme entre dos grafs G i H és una funció bijectiva φ entre els conjunts dels seus vèrtexs que preserva la relació d'adjacència.

$$\varphi : V(G) \to V(H)$$

Per tant, si un parell de vèrtexs $u, v \in G$ són adjacents en el graf G, llavors les seves imatges $f(u), f(v) \in H$ també hauràn de ser adjacents en el graf H. Si existeix un isomorfisme entre dos grafs G i H diem que els grafs són **isomòrfics**, el qual denotaríem com $G \simeq H$.

En el cas de grafs etiquetats, també es requereix que $l(v) = l(\varphi(v))$ per $\forall v \in V(G)$ i que $l((u,v)) = l((\varphi(u), \varphi(v)))$ per $\forall (u,v) \in E(G)$.

5.7 El problema d'isomorfisme de grafs

El problema d'isomorfisme de grafs consisteix en determinar si dos grafs són isomòrfics. Aquest és un problema NP, per tant tenim algorismes per determinar si dos grafs són isomòrfics o no, però fins el dia d'avui no s'ha trobat cap algorisme que resolgui aquest problema en temps polinòmic per a qualsevol parella de grafs.

Tot i això, sí que existeix un algorisme que pot determinar en temps polinòmic si dos grafs són isomòrfics en la gran majoria de casos: el test d'isomorfisme de Weisfeiler-Lehman.

Aquest problema ens interessa, ja que la xarxa neuronal haurà de ser capaç de diferenciar entre diferents tipus d'estructures de grafs per tal de classificar cada graf de forma correcta.

5.8 Test d'isomorfisme de Weisfeiler-Lehman

El test d'isomorfisme de Weisfeiler-Lehman és un algorisme que determina si dos grafs són isomòrfics, excepte en alguns tipus de grafs en concret, però de manera eficient.

El test produeix per cada graf la seva forma canònica (és a dir, és capaç de reduir els grafs a la seva representació més simple per tal de ser identificat de forma única). Si la forma canònica de dos grafs no és equivalent, llavors definitivament els grafs no són isomòrfics. En canvi, si la forma canònica de dos grafs és equivalent, no podem concloure que els dos grafs són isomòrfics, ja que és possible que dos grafs no isomòrfics tinguin la mateixa forma canònica (aquest és el motiu pel qual en alguns casos no es pot determinar si dos grafs són isomòrfics). Tot i això, els casos en els quals dos grafs poden compartir forma canònica són coneguts, llavors si es restringeixen els tipus de graf, també es pot arribar a afirmar que dos grafs sí que són isomòrfics. En el paper [6] es determinen quines són aquestes restriccions.

5.8.1 Com funciona el test WL

L'algorisme del test de WL utilitza el que s'anomena procediment de *colour refinement*: 1) inicialitza tots els vèrtexs al mateix color 2) Dos vèrtexs v, w canvien de color si existeix un color c per tal que v i w tinguin un nombre diferent de veïns de color c. El pas 2) es repeteix fins que la coloració es manté estable; i s'atura si el nombre de vèrtexs en un color determinat és diferent en els dos grafs.

Direm que la coloració es manté estable quan tots els vèrtexs del mateix color tenen el mateix nombre de veïns amb un color diferent.

Llavors, si els dos grafs que volem comparar tenen una coloració diferent, podem assegurar que els dos grafs no són isomòrfics. En canvi, si la coloració és la mateixa, els dos grafs són possiblement isomòrfics (el test WL és una condició necessària però no suficient per afirmar que són isomòrfics).

5.8.2 Versió unidimensional

També anomenada *color refinement*, simplement computa la coloració dels vèrtexs del graf.

Aquesta forma del test WL és similar a l'agregació de neighbours en les GNN (Graph Neural Networks) que s'explica més endavant.

5.8.3 Versió k-dimensional

Coloreja subgrafs, no només únicament vèrtexs, en aquest cas colorejem agrupacions de vèrtexs, així podem arribar a reconèixer estructures més complexes.

5.9 El problema de la semblança de grafs

El problema de la semblança de grafs, també anomenat *problema d'isomorfisme aproximat de grafs* o *problema de la coincidència de grafs*, intenta mesurar quant de similar és un graf a un altre, el que anomenem més formalment com a mesurar la distància entre grafs.

No hi ha una manera òbvia de mesurar la distància entre dos grafs, i per tant s'han proposat diferents formes de mesurar-la [7].

Aquest problema l'hauria de resoldre el model de xarxa neuronal en aquest treball, perquè el problema de l'isomorfisme permet determinar si dues converses són equivalents, però el més probable és que en els grafs d'entrada al model, que codifiquen les converses, no hi haurà cap isomorfisme (sobretot tenint en compte les etiquetes dels vèrtexs). El que hi haurà és un conjunt amb grafs que estaran a diferents distàncies entre ells, i caldrà una manera de determinar aquesta distància, o d'altra banda, caldrà fer simplificacions dels grafs originals per llavors trobar isomorfismes en els grafs simplificats.

Capítol 6

Teoria de ML i Xarxes neuronals

6.1 Teoria bàsica de Machine Learning

6.1.1 Flux de treball en aprenentatge automàtic

El flux de treball en aprenentatge automàtic consisteix a recopilar dades disponibles, netejar / preparar les dades, construir models, validar-los i desplegar el millor a producció.

- Fase de Training: S'utilitza el trainingset. S'introdueixen les dades al model i s'entrena, combinant l'entrada amb la sortida esperada.
- Fase de Testeig: S'estima com d'entrenat està el model (que depèn de la mida de les dades, del valor a predir, introduir, etc.) i es decideixen les propietats del model. La fase de Testeig sovint es divideix en dues parts:
  - 1) S'utilitza el validation set. S'observa l'error en la predicció del conjunt de possibles models a considerar, i es van actualitzant els hiperparàmetres dels models. En aquest pas es pot utilitzar cross-validation. Tot i això, en aquest pas la mitjana de les precisions de cada fold serà una precisió aproximada del model, caldrà la seguent fase per tenir una precisió més realista.
  - 2) S'utilitza el testset. Una vegada s'ha triat el millor dels models, llavors s'estima la precisió real del model triat amb dades no vistes encara pel mode. No es recomana que el testset siguin dades utilitzades en la cross-validation. Aquest és el moment on es pot aplicar *Early stopping*.

Una forma senzilla de veure-ho és amb el pseudocodi següent:

```
mentre (error en el validation set > x) {
    modificar hiperparametres
```

```
    mentre (error en el training set > y) {
        modificar parametres
    }
}
```

6.1.2 Datasets

La paraula *dataset* es refereix a un conjunt de dades (exemples). En qualsevol model de machine learning calen dades de les quals cal aprendre, però les dades poden ser de molts tipus. Cada exemple pot consistir en un simple valor enter, una imatge, una sèrie temporal, o en el cas d'aquest treball, la representació d'una conversa com un graf. Això si, cal definir el model de dades tal que cada exemple d'un dataset tingui el mateix format.

En machine learning sempre cal dividir el dataset complet en dues parts, el dataset d'entrenament (que anomenem *trainingset* o *datasetdetraining*) i el dataset de testeig (que anomenem *testset* o *datasetdetest*). Aquesta divisió es deu a que hi ha d'haver una part dels exemples que serveixin exclusivament per validar el model ja entrenat, degut que si utilitzem el mateixos exemples amb els quals hem entrenat el model, no podrem comprovar que el model ha generalitzat prou com per donar resultats correctes amb exemples que no ha vist mai. Per tant, el trainingset seran exemples que s'utilitzaràn exclusivament en la fase d'entrenament del model i el testset seran exemples que s'utilitzaràn exclusivament en la fase de validació (testeig) del model.

La mida en què cal dividir el dataset complet entre trainingset i testset no està definida. Com a guia, una de les divisions més comunes és utilitzar un 80% pel trainingset i un 20% pel testset. Recordar que no poden haver-hi mai exemples en comú entre trainingset i el testset, és a dir, la intersecció dels dos conjunts d'exemples ha de ser sempre el buit.

6.1.3 Cross-validation

La mètrica per la qual es guia l'optimitzador en la fase de validació dels models és la precisió (accuracy), i tal com s'ha implementat en aquest treball, només és fixa en el trainingset. Llavors, la fase final de testeig amb el testset es fa a part.

En el cas de la cross-validation, puc arribar a comparar models (fer la validació) només utilitzant el trainingset i no el testset, ja que s'utilitzen totes les dades del trainingset tan per entrenar com per validar el model. Això és possible dividint el trainingset en k parts iguals i completant les fases d'entrenament i validació k vegades. D'aquesta manera, en cada una de les k iteracions una de les parts s'utilitzarà exclusivament per la validació i la resta de parts seran exclusivament per l'entrenament. En cada una de les k iteracions es guarda la precisió del model, i al final es fa la mitjana aritmètica de les k precisions, tenint una precisió que ens permet saber la capacitat de generalització del model, sense haver de definir un dataset de validació manualment.

Les k particions del trainingset han de ser almenys dues ($k = 2$), i com a màxim el nombre d'exemples del dataset ($k = |dataset|$). Una de les particions més comunes és l'anomenada *10-fold-cross-validation*, on $k = 10$, i ho és perquè d'aquesta manera es té un 10% del dataset per a validar i el 90% restant per entrenar en cada una de les 10 iteracions.

En la figura 6.1 es mostra com el dataset complet es divideix en trainingset i testset, i llavors es fa un *5-fold-cross-validation* amb les dades del trainingset, permetent la comparació de models amb només el trainingset, i deixant el testset exclusivament per l'evaluació final del millor model ja escollit.

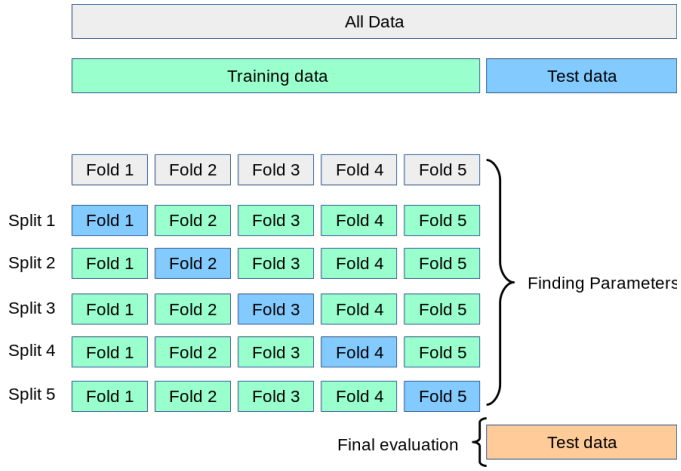

Figura 6.1: Divisió del dataset en 5-fold-cross-validation

L'inconvenient clar d'aquest mètode és que al passar de fer una sola iteració de n epochs, es passa a fer k iteracions de n epochs, multiplicant el temps de processat per k.

Així doncs, només utilitzant una part del dataset total (el trainingset) ja en tenim suficient per comparar diversos models, veient en cada cas la seva capacitat de generalització i una aproximació de la precisió de cada model. Com ens guardem el tetset com un conjunt de dades que cap model ha vist abans, fa la cross-validation bona mesura preventiva contra l'overfitting.

Aquests han sigut els resultats d'aplicar *5-fold-cross-validation* al model, és a dir, s'ha dividit el dataset de 6395 exemples en 5 parts iguals (és una divisió justa, 5116 exemples per entrenar i 1279 per validar), utilitzant els hiperparàmetres que millors resultats han donat:

S'ha fet una mitjana aritmètica de la precisió durant els 600 epochs de cada iteració,

També s'ha provat l'anomenat *stratified k-fold*, el qual és exactament igual al ja expli-

cat, però intenta que en cada fold (iteració) es preservin el percentatge d'exemples de cada classe. En el cas d'aquest treball, com el dataset no està balancejat, es provarà d'aplicar també la versió *stratified*. En la figura 6.2 es mostra com en un dataset amb 2 classes no balancejades, es divideixen les dades en folds seguint la distribució del dataset original.

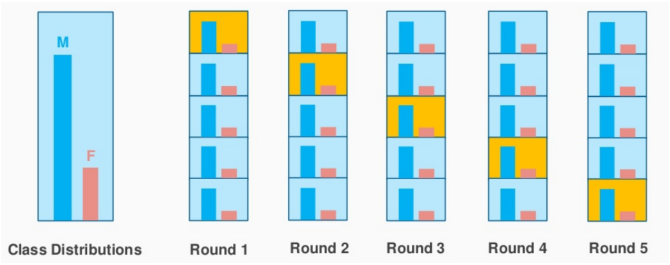

Figura 6.2: Divisió del dataset en stratified-5-fold-cross-validation

6.1.4 Early stopping

L'*early stopping*, tal com denota el seu nom, consisteix en parar el procés d'entrenament de manera prematura, abans d'acabar el nombre d'epochs que s'hagi determinat, ja que es considera que continuant no es milloraria el model.

Una mètrica fàcil de monitorar durant l'entrenament per tal de disparar l'*early stopping* és la pèrdua (és a dir, el valor de la *loss function* en un moment determinat). El problema és que la pèrdua no sempre captura el que més ens interessa sobre el que volem aconseguir amb el model.

En aquest treball s'ha utilitzat l'*accuracy* com a mètrica per decidir quin model és el millor, i per tant, serà aquesta mateixa mètrica la que servirà per disparar l'*early stopping*, donant per acabat l'entrenament.

Per tant, hem determinat que es dispari l'*early stopping* quan l'*accuracy* d'un model en el testset arriba a un cert percentatge d'encerts. No s'hauria d'utilitzar l'*early stopping* juntament amb la cross-validation, si es vol fer anar durant el procés de validació i no exclusivament en el testeig final amb el testset, s'ha de fer una validació dels models sense repeticions.

Si, en canvi, utilitzéssim la pèrdua com a mètrica, s'hauria de disparar l'*early stopping* quan la pèrdua deixa de decaure de manera important, estabilitzant-se.

6.2 Teoria bàsica de xarxes neuronals

Les xarxes neuronals reben una entrada (un sol vector, anomenat *input layer*) i el transformen a través d'una sèrie de *hidden layers*. Cada *hidden layer* (cada oculta) està

format per un conjunt de *units* (neurones), on cada neurona està totalment connectada a totes les neurones de la capa anterior i on les neurones d'una sola capa funcionen de forma totalment independent i no comparteixen cap connexió. L'última capa completament connectada s'anomena *output layer*. En un problema de classificació, cada neurona de l'*output layer* representa una única classe, havent-hi per tant el mateix nombre de neurones en aquesta capa que de possibles classes en la classificació.

6.2.1 Unit (neurona)

Una *unit* (neurona) sovint es refereix a la funció d'activació d'una capa mitjançant la qual les entrades es transformen mitjançant una funció d'activació no lineal (per exemple, mitjançant la funció *sigmoid*, explicada en l'apartat 7.4.2). Normalment, una neurona té diverses connexions entrants i diverses connexions sortints. Tanmateix, les unitats també poden ser més complexes, com les unitats de memòria a curt termini (LSTM), que tenen múltiples funcions d'activació amb un disseny diferent de connexions a les funcions d'activació no lineals o unitats de sortida màxima, que calculen la sortida final sobre una matriu de valors d'entrada transformats no linealment. Normalment, les funcions d'agrupació, convolució i altres funcions de transformació d'entrada no s'anomenen unitats.

6.2.2 Layers

Un *layer* (capa) és el bloc de construcció de més alt nivell en l'aprenentatge profund. Una capa és un contenidor que sol rebre una entrada ponderada, la transforma amb un conjunt de funcions majoritàriament no lineals i després passa aquests valors com a sortida a la següent capa. Una capa sol ser uniforme, és a dir, només conté un tipus de funció d'activació, agrupació, convolució, etc., de manera que es pot comparar fàcilment amb altres parts de la xarxa. La primera i l'última capa d'una xarxa s'anomenen capes d'entrada i sortida, respectivament, i totes les capes entremig s'anomenen capes ocultes.

6.2.3 Perceptró

Un perceptró és un simple algorisme de classificació binària que divideix un conjunt finit de senyals d'entrada de forma binària (0 o 1). A diferència de molts altres algorismes de classificació, el perceptró es va modelar segons la unitat essencial del cervell humà: la neurona.

Una combinació de perceptrons té capacitat per resoldre problemes complexos.

A la imatge 6.3 es veu la reprentació d'un perceptró. Les x_i són les entrades, els w_i són els pesos amb els *bias*, la Σ representa l'agregació de les entrades, la φ representa la funció d'activació i finalment la o representa la sortida.

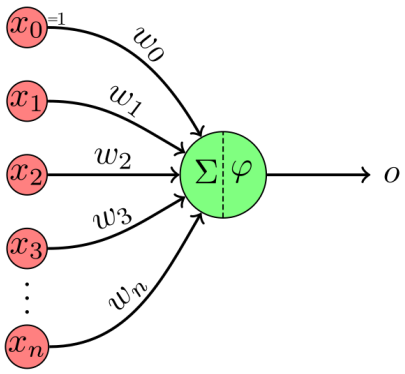

Figura 6.3: Perceptró

6.2.4 Perceptró multicapa (MLP)

Un perceptró multicapa (MLP) és un perceptró que s'uneix amb perceptrons addicionals, apilats en diverses capes, per resoldre problemes complexos.

La imatge 6.4 mostra un MLP amb tres capes. La primera capa (de color vermell) és la capa d'entrada, on anirien els valors que es volen entrar al MLP, la segona és una capa oculta, on els perceptrons reben com a entrada la sortida de tots els perceptrons de la capa anterior, i l'última capa (de color verd) és la capa de sortida, que també rebria el resultat dels perceptrons de la capa anterior i denotaria el valor de sortida del MLP.

Un MLP de tres capes, tal com s'ha mostrat en la imatge 6.4, es considera un MLP no profund. Si afegissim una altra oculta, tenint quatre capes, ja es consideraria un MLP profund (també anomenat xarxa neuronal profunda). Les capes d'entrada i sortida són obligatòries, i només es consideraria com a xarxa neuronal profunda si el nombre de capes ocultes és més gran que 1.

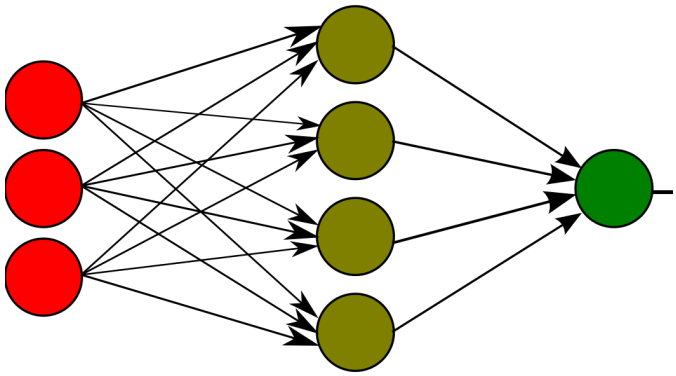

Figura 6.4: Perceptró multicapa

6.3 Elements d'una xarxa neuronal

6.3.1 Paràmetres

Els paràmetres són els coeficients del model, i són escollits pel propi model. Això significa que l'algorisme d'optimització, mentre aprèn, optimitza aquests coeficients (d'acord amb una determinada estratègia d'optimització) i retorna una matriu de paràmetres que minimitzen l'error. Per posar un exemple, en una tasca de regressió lineal, el model tindrà l'aspecte de $y = b + ax$, on b i a seran els paràmetres. L'únic que s'ha de fer amb aquests paràmetres és inicialitzar-los.

6.3.2 Hiperparàmetres

Els hiperparàmetres són elements que, a diferència dels paràmetres, els hi cal assignar un valor. El model no actualitzarà aquests valors segons l'algorisme d'optimització, sinó que caldrà canviar-los manualment segons els resultats de la validació del model ja entrenat.

6.4 Optimizers

6.4.1 Què és un optimizer

Un *optimizer* és un algorisme o mètode utilitzat per canviar els atributs de la xarxa neuronal com els pesos W o el learning rate, per tal de millorar el model. Els algorismes resolen problemes d'optimització que tenen per objectiu minimitzar la loss function, que s'explica a continuació.

6.4.2 Loss functions

La loss function (també anomenada funció de pèrdua, funció de cost o funció d'error) és una funció que ha de reflectir tots els possibles bons o mals aspectes d'un model en un sol valor real. Si podem quantificar el bon o mal comportament d'un model en concret, llavors es podran comparar els possibles models candidats i escollir-ne el millor.

En el nostre cas, un problema de classificació de grafs amb la seva polarització, el valor serà la penalització per una classificació incorrecta d'un graf amb la seva polarització. Com més grafs incorrectament classificats més alt serà el valor.

És necessari escollir una loss function pel nostre model, ja que estem davant un problema d'optimització que busca minimitzar el nombre de grafs classificats incorrectament, és a dir, minimitzar la loss function.

Hi ha diverses loss functions, però les més utilitzades en problemes de classificació multiclasse són la funció *cross-Entropy* i la funció *negative log likelihood*. La funció *negative log likelihood* és l'escollida en aquest treball.

En aquest apartat cal una apreciació. Perquè l'optimizer arribi a un mínim local en el nostre espai de pesos W (funció de pèrdua), caldrà una guia a seguir que indiqui una direcció cap al mínim local. No fa falta recórrer a direccions aleatòries, ja que existeix una manera que garanteix matemàticament arribar a un mínim local: el gradient.

6.4.3 Gradient

Existeix una forma que ens permet calcular la millor direcció a seguir per arribar a un mínim local (es garanteix que dona per resultat la baixada més pronunciada). Aquesta direcció estarà relacionada amb el gradient de la funció de pèrdua.

De manera intuitiva, es pot pensar el gradient com a sentir el pendent d'un turó per sota dels nostres peus i baixar per la direcció que se sent més forta.

En funcions unidimensionals, el pendent és la velocitat instantània de canvi de la funció en qualsevol punt en concret. El gradient és una generalització del pendent per a funcions que no prenen un sol nombre sinó un vector de nombres. A més a més, el gradient és només un vector de pendents (és a dir, derivades) per a cada dimensió de l'espai d'entrada. L'expressió matemàtica per a la derivada d'una funció unidimensional respecte a la seva entrada és:

$$\frac{df(x)}{dx} = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$

Quan les funcions prenen un vector de nombres en lloc d'un sol nombre, anomenem a les derivades *derivades parcials*, i el gradient és simplement el vector de derivades parcials en cada dimensió.

6.4.4 Gradient descent

Gradient descent és un algorisme d'optimització que tracta de minimtzar la funció de pèrdua movent-se iterativament en la direcció definida pel negatiu del gradient.

Per trobar un mínim local de la funció de pèrdua mitjançant el descens del gradient, fem passos proporcionals al negatiu del gradient (o un gradient aproximat) de la funció en el punt actual.

Es fa el primer pas cap avall en la direcció especificada pel desnivell negatiu (negatiu del gradient de la funció de pèrdua, tenint en compte els paràmetres inicials de la xarxa). El nou punt on s'arriba consisteix en els paràmetres inicials de la xarxa actualitzats pel gradient negatiu. A continuació, tornem a calcular el gradient negatiu (passant les coordenades del nou punt, és a dir, els paràmetres actualitzats de la xarxa) i fem un altre pas en la direcció que especifica. Continuem aquest procés de manera iterativa fins que arribem al final del nostre gràfic o fins a un punt en què ja no ens podem moure cap avall, és a dir, on hi ha un mínim local.

El tipus més bàsic de gradient descent és el Batch Gradient Descent. Es calcula el gradient de la loss function en cada pas pels paràmetres de tot el trainingset sencer, per cada epoch, el qual és molt exagerat, fins i tot intractable. Per aquest motiu, es diferencien tres tipus de gradient descent segons el nombre d'exemples del trainingset utilitzats:

- Batch Gradient Descent. La mida del batch (veure apartat 7.5.5) és el total d'exemples del trainingset. Es sol utilitzar en la validació del model.
- Stochastic Gradient Descent. La mida del batch és 1. Poc utilitzat, ja que calcular el gradient pels paràmetres d'un sol exemple fa que la direcció calculada estigui totalment condicionada per un sol exemple, tenint direccions errònies i tardant molt més a arribar a un mínim local.
- Minibatch Gradient Descent. La mida del batch és més gran que 1 i menor al total d'exemples del trainingset. Es sol utilitzar en l'entrenament del model.

A vegades s'anomena de manera general *Stochastic Gradient descent (SGD)* encara que estiguem utilitzant minibatches. Llavors, hi ha tècniques per millorar el SGD, el qual resulten en algorismes d'optimització basats en gradient descent, però millorats per tal d'arribar de la forma més ràpida a un bon mínim local.

Alguns d'aquests algorismes millorats són:

- SGD amb *Momentum*: similarment al concepte de quantitat de moviment en mecànica clàssica, si una direcció es manté en diverses iteracions, guanya moment (velocitat), i si la direcció canvia, perd moment. Això fa que s'arribi més ràpid al mínim local. Hi ha una millora addicional anomenada Nesterov Accelerated Gradient, que s'anticipa observant el gradient per evitar que guanyi tanta velocitat que es passi el mínim local.

- AdaGrad: adapta el learning rate basant-se en els paràmetres de la xarxa, fent grans canvis quan els paràmetres són poc frequents, i pocs canvis quan són molt frecuents. Així hi ha un learning rate individual per cada paràmetre. També incorpora el Nesterov Accelerated Gradient.
- AdaDelta: incorpora les dues millores anteriors, però impedeix que el learning rate disminueixi continuament fins al punt que la xarxa deixi d'aprendre, com pot passar amb AdaGrad.
- Adam. incorpora tote les millores anteriors. Guarda el momentum per cada paràmetre, igual com es fa a AdaGrad amb el learning rate.

6.5 Arquitectures de xarxes neuronals més comunes

6.5.1 Convolució

La convolució és una operació matemàtica en dues funcions f i g (o peces d'informació) tal que es produeix una tercera funció $h = f * g$ que expressa com la forma d'una funció es modificada per l'altra.

El mapa de característiques (o dades d'entrada) i el nucli de convolució es combinen per formar un mapa de característiques transformat. La convolució s'interpreta sovint com un filtre, on el nucli filtra el mapa de funcions per obtenir informació d'un tipus determinat (per exemple, un nucli pot filtrar les vores i descartar altra informació).

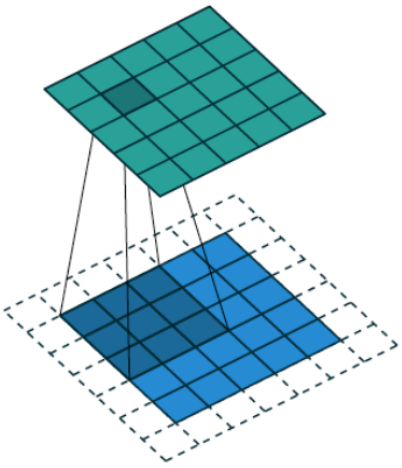

Figura 6.5: Visualització de l'operació de convolució

6.5.2 Pooling

El **pooling** és un procediment que pren informació sobre una àrea determinada i la redueix a un valor únic.

Una de les raons per aplicar pooling, reduint la "qualitat"de les dades, és que d'aquesta manera es podran cercar característiques més generalitzables en les dades. Aquesta reducció ajuda la capa de convolució que vingui després de la capa de pooling a buscar çaracterístiques de nivell superior".

El tipus més comuns són el *max pooling* i el *mean pooling*.

El *max pooling* es realitza aplicant un filtre màxim subregions no superposades de la representació inicial (és a dir, es queda amb el valor màxim d'una subregió). Això es pot observar en la imatge 6.6. El *mean pooling* només es diferencia del *max pooling* amb que s'aplica un filtre de mijana, és a dir, es queda amb la mitjana dels valors d'una subregió.

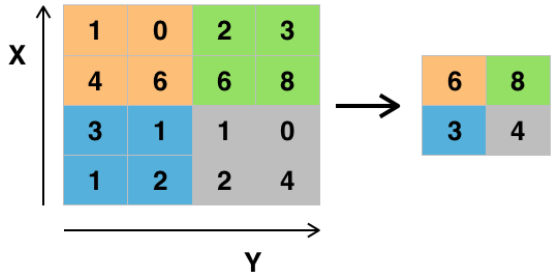

Figura 6.6: Max pooling

6.5.3 Xarxes neuronals convolucionals (CNN)

Una xarxa neuronal convolucional utilitza capes de convolució que filtren les entrades per obtenir informació útil. Aquestes capes convolucionals tenen paràmetres que s'aprenen per tal que aquests filtres s'ajustin automàticament per extreure la informació més útil per a la tasca actual. Per exemple, en una tasca general de reconeixement d'objectes pot ser molt útil filtrar informació sobre la forma d'un objecte (els objectes solen tenir formes molt diferents) mentre que per a una tasca de reconeixement d'ocells pot ser més adequat extreure informació sobre el color de la ocell (la majoria d'ocells tenen una forma similar, però colors diferents; aquí el color és més útil per distingir entre ocells). Les xarxes de convolució s'ajusten automàticament per trobar la millor característica per a aquestes tasques. Normalment, s'utilitzen múltiples capes de convolució que filtren les imatges per obtenir informació cada vegada més abstracta després de cada capa.

Les xarxes convolucionals també solen utilitzar capes de pooling per a una translació i rotació limitades (detectar l'objecte encara que aparegui en algun lloc inusual). L'a-

grupació també redueix el consum de memòria i, per tant, permet l'ús de més capes convolucionals.

El la imatge 6.5 es veu com seria una arquitectura de CNN utilitzant capes de convolució i de pooling.

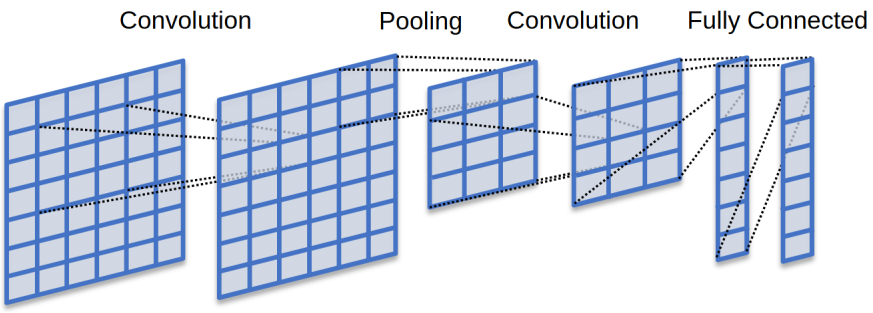

Figura 6.7: Convolució i Pooling en l'arquitectura d'una xarxa CNN

6.6 Xarxes neuronals per a grafs (GNN)

Aquesta és l'arquitectura de xarxa neuronal que utilitzarem.

Tractant grafs sense bucles (arbres), podríem simplificar l'arquitectura GNN, arquitectures anteriors a GNN basades en RNN (Xarxes neuronals recurrents) havien treballat amb arbres, però en aquest treball preferim generalitzar a grafs de qualsevol tipus i característiques per tal de seguir funcionant en grafs amb bucles, per exemple en el model de dades centrat amb l'autor.

Els grafs tenen una estructura arbitrària, no tenen ni un principi ni un final, són un conjunt de valors sense cap localitat específica en l'espai de dades. Són un conjunt de vèrtexs que es connecten entre ells amb arestes però això no implica que necessàriament que dos vèrtexs connectats estiguin aprop.

Les xarxes neuronals convencionals funcionen amb dades estructurades, com seqüències de text, sèries temporals o imatges. Per exemple, les imatges estàn estructurades per la quadrícula que formen els píxels, que sempre tenen la mateixa proximitat entre si.

El model de xarxes neuronals per a grafs, també anomenat GNN, dóna solució a aquest problema. En el primer paper en que es va definir aquest model[8] es va demostrar que donat un graf parcialment etiquetat G, una GNN podia arribar a predir de manera precisa les etiquetes que faltants del graf G.

El model GNN aprenia a representar cada vèrtex amb un vector d'estats que conté informació sobre els nodes adjacents. Per computar el resultat, el model passa aquest

vector d'estats en una funció de sortida.

Tot i això, la variant original té varies limitacions com que és computacionalment costosa o que no processava dades sobre les arestes. Per això, l'arquitectura de la xarxa neuronal en aquest treball haurà de ser més una extensió de la GNN més bàsica.

6.6.1 Definició formal

Sent $G = (V, E)$ un graf amb un vector de característiques de vèrtex X_v per vèrtexs $v \in V$. Hi han dues tasques a diferenciar:

- **Classificació de vèrtexs**: cada vèrtex $v \in V$ té associat un vector d'etiquetes y_v i s'ha d'aconseguir aprendre un vector de representació h_v de v tal que l'etiqueta del vèrtex v pugui ser predida com $y_v = f(h_v)$
- **Classificació de grafs**: donat un conjunt de grafs $G_1, ..., G_N$ i les seves etiquetes $y_1, ..., y_N$, s'ha d'aconseguir aprendre un vector de representació h_G que predigui l'etiqueta d'un graf sencer, $y_G = g(h_G)$

En aquest treball, com el problema tracta de classificació de grafs, la teoria es centrarà amb la variant per a classificació de grafs.

Mecanisme de pas de missatges

Per tant, les GNN utilitzen l'estructura del graf i les característiques del vèrtex X_v per aprendre a representar un vector de representació d'un vèrtex, h_v, o del graf sencer, h_G. Les GNN més actuals segueixen una estratègia d'agregació de veïnatge (*neighborhood aggregation strategy*), on s'actualitza iterativament la representació d'un vèrtex agregant les representacions dels seus veïns (vèrtexs adjacents). Després de k iteracions, la representació del vector ja capturarà la informació estructural de tots els vèrtexs veïns a una distància de k salts. Formalment, la capa k d'una GNN aconsegueix actualitzar les representacions de cada vèrtex amb les funcions següents:

$a_v^{(k)} = AGREGA^k(h_u^{k-1} : u \in N(v)),$

$h_v^{(k)} = COMBINA^k(h_v^{k-1}, a_v^k)$

on $h_v^{(k)}$ és el vector de característiques del vèrtex v en la iteració (capa) número k. Aquest vector s'inicialitza com $h_v^{(0)} = X_v$

i $N(v)$ és un conjunt de vèrtexs adjacents a v.

Aquest mecanisme, que de forma iterativa actualitza la representació de cada vèrtex del graf utilitzant les funcions $AGREGA$ i $COMBINA$, s'anomena mecanisme de pas de missatges (en anglès, **message-passing**).

L'elecció de les funcions $AGREGA^k(\cdot)$ i el $COMBINA^k(\cdot)$ són crucials en les GNN.

En el cas d'un problema de classificació de nodes, el vector de representació del vèrtex $h_v^(K)$ en la última iteració és el que s'utilitza per fer la predicció. En canvi, en el problema de classificació de grafs, cal una funció addicional $READOUT$ que agregui les representacions en la iteració final de cada vèrtex del graf obtenint la representació del graf sencer, h_G:

$h_G = READOUT(h_v^K | v \in G)$

La funció $READOUT$ ha de ser una funció de *pooling* a nivell de graf.

6.6.2 Alguns tipus específics de GNN

Sobretot cal tenir en compte quines arquitectures permeten problemes de classificació de grafs, i que permetin un graf arbitrari com a entrada (que permetin nodes i arestes amb pesos o sense, diferents ordres i mides de graf, etc.).

Com que l'arquitectura GIN (Graph Isomorphism Network) compleix totes les premises, serà l'escollida en aquest treball. A més es pot veure en diversos papers[9][10] com sol donar bons resultats en diferents problemes de classificació de grafs.

Existeix un ampli nombre d'arquitectures GNN. Per exemple, la GAT (Graph Attentention Network), la GCN (Graph Convolutional Network), la GraphSAGE, o la GIN.

És interessant distingir les GNN segons el seu pas de propagació (és a dir, segons la funció $AGREGA$ utilitzada). En la imatge 6.8 es pot veure aquesta classificació, on tant la GCN, la GraphSAGE i la GIN utilitzen un agregador convolucional. El concepte és similar a una xarxa neuronal convolucional. Per tant, aquestes xarxes funcionen amb dades d'imatges. La idea fonamental continua sent la mateixa. Les dades d'alt nivell, mitjançant convolucions, es converteixen en dades de mida inferior, amb conceptes més elementals. Llavors, dins d'aquesta categoria, es distingeix entre les *spectral-based convolutional GNNs* i les *spatial-based convolutional GNNs*.

La GAT, no entraria dins de cap d'aquestes dues categories, ja que no és de tipus convolucional.

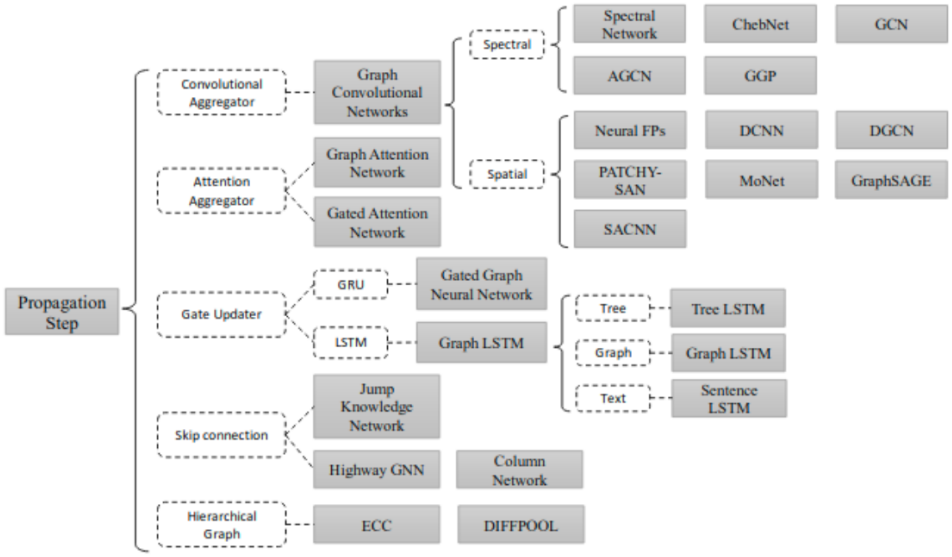

Figura 6.8: Vàries arquitectures de GNN segons el seu pas de propagació. Imatge extreta de Jie Zhou et al.

6.6.3 Xarxes neuronals d'isomorfisme de grafs (GIN)

L'arquitectura GIN està basada en el Test d'isomorfisme de Weisfeiler-Lehman per estudiar la capacitat expresiva de les GNN. En el paper original [10] els autor argumenten que l'arquitectura proposada fa que el model sigui possiblement tan potent com el test d'isomorfisme de Weisfeiler-Lehman.

Com s'ha mostrat en l'apartat 6.6.2 l'arquitectura GCN és la base de l'arquitectura GIN, però en la GIN l'estratègia d'agregació de veïnat és *spatial-based*, el qual es descriu com una millor manera de representar l'agregació de vèrtexs veïns.

- Esquema d'agregació

L'arquitectura GIN, al ser de tipus *spatial-based convolutional GNNs*, utilitza la funció $AGREGA$ següent:

$a_v^{(k)} = AGREGA^k(h_u^{k-1} : u \in N(v))$

$a_v^{(k)} = MAX(ReLu(W \cdot h_u^{k-1}), \forall u \in N(v))$

on W és una matriu que es pot aprendre i MAX representa *element-wise max-pooling* (*max-pooling* utilitzant el producte Hadamard).

- Capa de pas de missatges

Llavors, la funció *COMBINA* en el model GIN té la definició següent:

$h_v^{(k)} = COMBINA^k(h_v^{k-1}, a_v^k)$

$h_v^{(k)} = MLP^k((1 + e^k \cdot h_v^{(k-1)} + \sum_{u \in N(v)} h_u^{(k-1)}))$

on e^k és un paràmetre que es pot aprendre, i MLP és un *perceptró multicapa*.

Les convolucions espacials utilitzades en el model GIN són similars a les convolucions regulars en la manera que la convolució es fa en un espai determinat de les dades. Les convolucions generen un nou píxel a partir del mateix píxel (el píxel situat a la mateixa posició que el nou) i està envoltant píxels, les convolucions de grafs espacials fan convolucions agregant un node i els seus veïns en un nou node. Més concretament, les convolucions espacials fan una mitjana de les característiques dels vèrtexs adjacents a un node central, modificant aquest vèrtex central.

La imatge 6.9 il·lustra la diferència entre una quadrícula estructurada, com passa en una imatge, amb una convolució espacial en un graf.

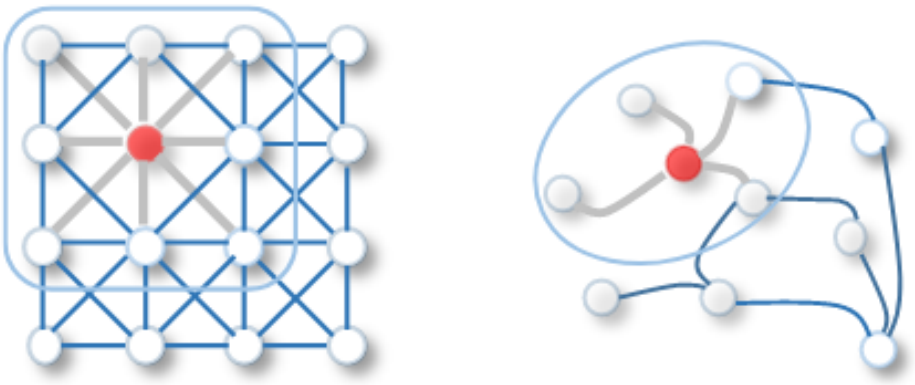

Figura 6.9: A l'esquerra, convolució 2D d'una imatge. A la dreta, convolució espacial d'un graf. Imatge extreta de Wu et al.

Cal recordar que no ens serviria utilitzar la convolució convencional en grafs. Els grafs no són quadrícules estructurades, sinó que els vèrtexs adjacents a un node no tenen un ordre i poden variar en nombre.

Capítol 7

Pràctica

7.1 Descripció del problema

En l'entrada de la xarxa neuronal: hi ha d'haver un graf, per tant com s'ha explicat en la part teòrica, estem restringits a utilitzar una arquitectura de GNN.

En la sortida de la xarxa neuronal: hi ha d'haver el valor de la polarització del graf d'entrada, per tant estem davant d'un problema de classificació, on es classifica un graf sencer (conversa) a un valor (polarització normalitzada de la conversa). S'ha decidit determinar el valor de la polarització utilitzant un decimal per no tenir un nombre molt elevat de possibilitats en la classificació. Amb un decimal, essent la polarització un valor entre -1 i 1, tenim 21 possibilitats $(-1, -0.9, ..., -0.1, 0, 0.1, ..., 1)$. Per tant, estarem davant d'un problema de classificació multiclasse (es pot classificar un graf en una de les 21 classes possibles).

7.2 PyTorch, PyTorchGeometric i CUDA

7.2.1 PyTorch

PyTorch és un paquet per dur a terme càlculs numèrics fent ús de la programació de tensors (els tensors vindrien a ser vectors de vàries dimensions). PyTorch es sol utilitzar en l'investigació i desenvolupament dins de l'aprenentatge automàtic, i està centrat principalment en xarxes neuronals.

PyTorch té una interfície molt senzilla d'utilitzar per a crear xarxes neuronals tot i treballar de forma directa amb tensors, i no es necessita una llibreria de nivell superior com podria ser Keras o Tensorflow.

PyTorch disposa de suport per a executar-se en targetes gràfiques (GPU), ja que internament utilitza CUDA.

Per tant, aquesta llibreria ens serà necessària per tal de desenvolupar i entrenar el model de xarxa neuronal.

7.2.2 PyTorchGeometric

PyTorch Geometric és una llibreria d'aprenentatge profund geomètric per a PyTorch.

Consisteix en diversos mètodes per a l'aprenentatge profund en grafs i altres estructures irregulars, també coneguts com a aprenentatge profund geomètric, a partir de diversos articles publicats. A més, incorpora un carregador de *mini-batches* per a grafs de qualsevol mida, un gran nombre de conjunts de dades de referència comuns (basats en interfícies senzilles per crear el vostre) i transformacions útils, tant per aprendre en gràfics arbitraris, com en malles 3D o núvols de punts.

Un d'aquests mètodes per a l'aprenentatge profund geomètric a partir de diversos articles publicats que conté aquesta llibreria és justament l'arquitectura GIN tal com s'explica en el paper original aquesta arquitectura ja anomentat[10]. La llibreria conté una implementació de l'esquema d'agragació (funció *AGREGA*) i de la capa de pas de missatges (funció *COMBINA*) tal i com es defineix en el paper, per tant, ja no caldrà implementar aquestes operacions manualment. Es pot trobar la implementació de les dues funcions en el repositori de GitHub de la llibreria.

A més, l'arquitectura GIN té una extensió anomenada GINE per si es requereix introduir grafs amb arestes etiquetades (amb pesos), i no només vèrtexs etiquetats. La implementació d'aquesta variant està en el mateix fitxer que l'implentació de les dues funcions de GIN.

7.2.3 CUDA

Per entrenar la xarxa neuronal es pot utilitzar tan la CPU com la GPU.

Si s'utilitza la GPU, es pot fer ús de la tecnologia CUDA. Amb aquesta tecnologia es pot accelerar l'execució fent servir unitats de procesament gràfic amb capacitat CUDA (i als seus elements de comput paral·lel). Com que les operacions d'àlgebra lineal que es duen a terme durant l'entrenament de la xarxa s'executen en paral·lel, es pot arribar a disminuir el temps d'entrenament 100 vegades.

PyTorch integra la possibilitat d'utilitzar operacions de CUDA de forma nativa. Simplement cal seleccionar la GPU a utilitzar (òbviament cal que sigui una GPU amb tecnologia CUDA i estar prèviament configurada), i indicar les dades que cal carregar a la GPU per tal de fer operacions amb elles.

7.2.4 InMemoryDatasets

InMemoryDataset és una classe abstracta de PyTorchGeometric que permet definir datasets que es poden carregar en memòria RAM (o la VRAM si s'utilitza CUDA i es carreguen les dades a la GPU).

Creant una subclasse de InMemoryDatasets que implementi els mètodes d'aquesta classe podrem carregar de forma senzilla les dades del dataset utilitzant un el *Dataloader*.

Dataloader és el carregador de dades de PyTorchGeometric, i carrega dades del tipus *Dataset*. Això permet que pugui carregar datasets en memòria no volàtil com en memòria volàtil. Com la classe abstacta InMemoryDataset hereda de la classe Dataset, una vegada implementada la subclasse de InMemoryDataset ja tindrem instàncies que podràn ser carregades.

Per més informació sobre els datasets amb PyTorchGeometric consultar. la seva documentació oficial

El codi de la implementació del dataset es pot veure en el script de nom $reddit_dataset.py$.

7.3 Format i processat de les dades

En la fase d'entrenament, es divideix el dataset en el trainingset, dades les quals s'utilitzaràn exclusivament per entrenar la xarxa, i el testset, unes dades que no s'utilitzen mai per entrenar la xarxa, sinó que serveixen exclusivament per comprovar com de precís és la xarxa neuronal en cada moment.

7.4 Paràmetres

Tal com s'ha explicat en l'apartat 6.3.1, els anomenats *paràmetres* de la xarxa tenen un sol possible valor d'acord amb el problema a resoldre, o simplement són una decisió de disseny que cal haver pres abans de començar a entrenar la xarxa per primera vegada, i per tant es solen deixar com a definitius. Aquest seran els casos que s'expliquen en aquesta secció.

7.4.1 Input i output layers

La mida de la capa d'entrada i la de sortida és un paràmetre. Tal com ja s'ha explicat en altres apartats, la capa d'entrada estarà formada per la representació del graf i la mida és igual al nombre de característiques d'un vèrtex, el qual és 2 (score i sentiment). La mida de la capa de sortida és de 21, ja que aquest és el nombre de classes diferents en el problema de classificació.

7.4.2 Funcions d'activació

Hi ha diversos tipus de funcions d'activació. Un possible tipus és una funció d'activació lineal. Aquesta possibilitat no és viable en el nostre cas, ja que ens limita molt el nostre model. Això és deu al fet que no es pot utilitzar backpropagation (gradient descent) per entrenar el model, ja que la derivada d'una funció lineal és una constant. No importaria el nombre de capes introduïdes, ja que la composició de dues funcions lineals és una

funció lineal. No estaríem computant funcions més interessants, encara que s'estigui profunditzant en la xarxa.

En definitiva, el nostre model de xarxa es veuria reduit a un simple model de regressió lineal, el qual seria clarament insuficient per tractar l'alt nivell de complexitat del model de dades.

Es recomana la funció *sigmoid* en cas de clasificació binària, si no és el cas una altra seria més recomanable. En el cas d'aquest treball, s'ha escollit la funció d'activació ReLu, ja que així ho indica en el paper original del model GIN[10].

En la ReLu $g(z) = max(0, z)$.
La derivada $g'(z)$ val:

- valdrà "0"si $z < 0$
- valdrà "1"si $z > 0$
- valdrà "no definit"si z = 0. Però com que el cas que z = 0.000...0 és molt baix i no podem tenir un valor no definit, fem que valgui "1"quan $z \geq 0$

7.5 Hiperparàmetres de la xarxa neuronal

En aquesta secció, contrariament a la secció anterior, s'expliquen els hiperparàmetres (veure secció 6.3.2, els quals no estàn determinats abans de començar a entrenar la xarxa neuronal.

En aquests casos, s'utilitza el prova i error, que encara que sigui un mètode clarament ineficient, és molt utilitzat a l'hora de determinar aquests paràmetres. Això es deu a que no hi sol haver un procediment que determini quin ha de ser el valor del paràmetre donades les característiques del problema a resoldre i el model de xarxa neuronal, sinó que simplement caldrà provar diversos valors i quedar-nos amb els valors dels paràmetres que maximitzin els encerts (la precisió) de la nostra xarxa.

7.5.1 Nombre d'epochs

El nombre d'epochs és un hiperparàmetre el qual defineix el nombre de vegades que l'algorisme d'aprenentatge processarà la totalitat del dataset de training. És un paràmetre que cal especificar obligatòriament abans d'entrenar la xarxa.

El nombre mínim d'epochs és 1 (almenys cal processar una vegada cada exemple del nostre trainingset), i no hi ha un màxim establert. Tan es pot definir un nombre fix d'epochs i acabar l'entrenament de la xarxa una vegada s'ha completat tots els epochs; com parar l'entrenament abans que el límit d'epochs fixat segons el comportament de la xarxa, utilitzant l'anomenat *early stopping* (mirar apartat 6.1.4).

Si no s'utilitza un nombre suficient d'epochs hi haurà underfitting (no haurà pogut aprendre suficientment les dades del trainingset), però si una vegada s'ha arribat al

mínim local es continuen fent més epochs, hi haurà overfitting (processant les mateixes dades un nombre excessiu de vegades provoca que aprengui massa bé el trainingset i que el model perdi capacitat de generalitzar).

Això es pot comprovar en les figures 7.1 i 7.2, on es mostra com en els primers epochs la funció de pèrdua encara no s'ha minimitzat prou i la accuracy és baixa, però a partir del epoch 600 (aproximadament) la funció de pèrdua decau molt més lentament, la accuracy del trainingset augmenta significativament però la del testset es manté o baixa, un signe clar de overfitting. Ens hem de quedar en el punt on l'accuracy del testset és més alta (i la accuracy del trainingset no ha augmentat molt més que la del testset).

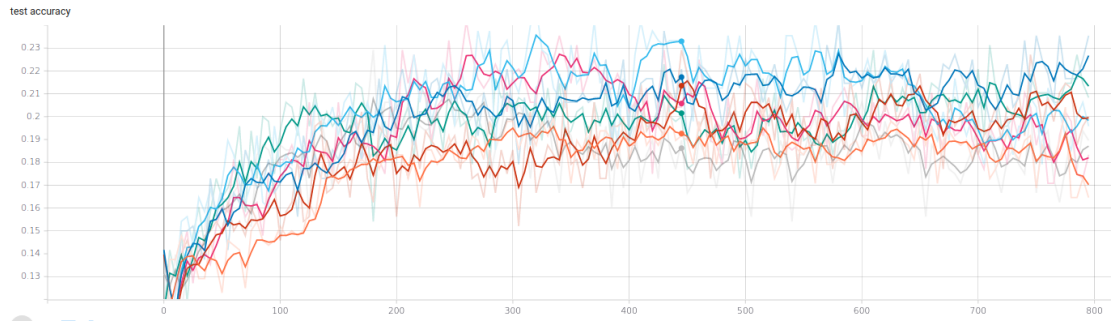

Figura 7.1: Precisió en el testset durant 800 epochs en el dataset de 6395 exemples

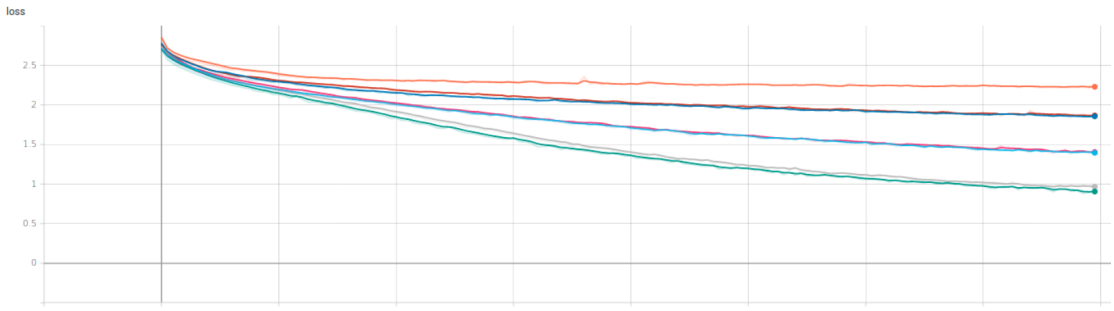

Figura 7.2: Valor de la funció de pèrdua durant 800 epochs en el dataset de 6395 exemples

Continuar entrenant la xarxa després de que passin un nombre cosiderable de epochs amb la funció de pèrdua més o menys estable no aportarà cap millora al model. En aquest exemple s'ha escollit un nombre excessivament alt de epochs per veure el comportament del model en aquest cas. Per tant, una vegada la funció de pèrdua és clarament estable l'entrenament de la xarxa hauria d'acabar.

7.5.2 Nombre de neurones (unitats) per capa oculta (dimensió d'una capa oculta)

El nombre de neurones per capa oculta s'ha de determinar per prova i error. Normalment s'utilitzen potències de dos (16, 32, 64, etc.).

7.5.3 Nombre de capes ocultes

El nombre de capes ocultes s'ha de determinar per prova i error.

No sempre més capes ocultes implicarà que augmenti la precisió. Tot i que la xarxa té més capacitat per aprendre, una xarxa excessivament gran aprendrà més el soroll de les dades (propietats específiques de cada exemple no generalitzables), en canvi, si s'elegeix el tamany just, aprendrà solament les propietats que són generalitzables i no tindrà lloc per aprendre les especificitats de cada exemple. Augmentar la dimensió d'una capa oculta té les mateixes implicacions.

7.5.4 Dropout

El *dropout* és un mètode de regularització que redueix l'overfitting de les dades. El mètode consisteix en obviar alguns valors de sortida de les capes durant l'entrenament, simulant que cada capa té un nombre diferent d'unitats i una connectivitat diferent en cada actualització. El valor del dropout és un real entre 0 i 1, indica el percentatge de valors a ometre.

7.5.5 Mida del batch

La mida del batch és un hiperparàmetre del gradient descent que defineix el nombre d'exemples que s'utilitzaràn abans d'actualitzar el paràmetres interns del model. Quan s'arriba a l'últim exemple d'un batch, les prediccions en cada un dels exemples es comparen amb els outputs correctes i es calcula l'error, el qual es propaga (tal com s'ha explicat en backpropagation, rebaixant el valor del gradient d'error).

La mida del batch ha de ser $1 <= midadelbatch <= nombred'exemplesdeltrainingset$. El més recomanable és escollir un valor major a 1 però que no arribi a ser el nombre total d'exemples.

Hi ha un $tradeoff$ entre el fet d'augmentar o disminuir la mida del batch. - Quan s'augmenta la mida del batch, a l'utilitzar més valors per calcular el gradient i actualizar el model, s'aconseguirà que es segueixi una direcció més adequada per arribar a un bon mínim local en la funció de pèrdua. Però la conseqüència d'utilitzar més exemples en el batch és que caldrà utilitzar més memòria i caldrà més temps per actualitzar el model.

- Quan es disminueix la mida del batch, a l'utilitzar menys valors per calcular el gradient i actualizar el model s'incrementarà el soroll del procés d'entrenament, ja que els exemples individuals influiran més en direcció que es seguirà en la funció de pèrdua. Tot i això, si

hi ha una limitació en la memòria o en el temps de processat, caldrà disminuir la mida del batch forçosament.

7.5.6 Learning rate

El learning rate és un dels paràmetres que poden millorar més els resultats si d'optimitzen. El learning rate determina la quantitat de canvi que hi haurà en el model en cada epoch.

Com s'ha explicat anteriorment, durant el pas de backpropagation es canvien els pesos es propaga l'error a cada node de la xarxa, doncs en el cas d'utilitzar el learning rate, no s'actualitzen els pesos amb el valor d'error total, sinó que es multiplica el valor de l'error pel learning rate. Aíxi doncs, amb un learning rate igual a 0 els pesos de la xarxa no canviarien, i amb un valor de 1, canviarien totalment segons el valor de l'error, és a dir, com menor sigui el learning rate, menys canviarà el model.

El learning rate pot ser qualsevol valor decimal entre 0 i 1. Normalment s'utilitza un valor de learning rate de 0.1 en l'inici, que no s'acostuma a disminuir fins a menys de 10^{-6}.

Com que el model després de múltiples epochs tendeix al overfitting, és una bona idea que el model es vegi fortament influenciat pel l'error estimat que es propaga en l'inici de la fase d'entrenament, però que al llarg de l'entrenament es vagi disminuint, ja que s'apropa més al mínim local i canviar el model totalment continuament evita que s'estabilitzi.

Això es pot aconseguir en PyTorch utilitzant el Scheduler, el qual cada n epochs disminueix el learning rate segons el factor de decaïment (decay rate). Per exemple, un decay rate de 0.5, farà que el learning rate sigui la meitat que en l'anterior.

Tot i això existeix una manera més efectiva de modificar el learning rate, d'adaptant-lo millor als resultats de la xarxa durant l'entrenament. Aquest és el cas dels algorismes d'actualització, dels quals nosaltres hem utilitzat l'algorisme d'actualització Adam.

7.5.7 Algorisme d'actualització Adam

El rendiment del nostre model en el dataset de training pot ser monitoritzat per un algorisme d'aprenentage adaptatiu, el qual ajusta el learning rate segons aquests resultats. En aquest cas diríem que estem utilitzant un learning rate adaptatiu.

Un learning rate adaptatiu generalment millora el rendiment d'un model comparantho amb un learning rate fix (aplicant-hi opcionalment un factor de decaïment definit per l'usuari), ja que definint-lo a priori per l'usuari només tenim de guia el prova i error.

Tal com explica el paper en el que es basa la implementació de l'algorisme Adam de PyTorch (l'utilitzat en aquest treball), és compatible amb el scheduler utilitzat per re-

baixar el *learning rate* segons el comportament del model. En aquest cas es veu necessari afegir el scheduler ja que durant l'entrenament no es veu que l'algorisme Adam canvii el learning rate en cap moment, el qual podria ser perquè en el nostre cas no s'arriben a complir les condicions necessàries per a modificar el learning rate.

Capítol 8

Arquitectura del model GIN implementat

8.1 Diagrama

En la figura 8.1 es pot veure el diagrama de l'arquitectura del model GIN definitiu que s'ha definit en aquest treball.

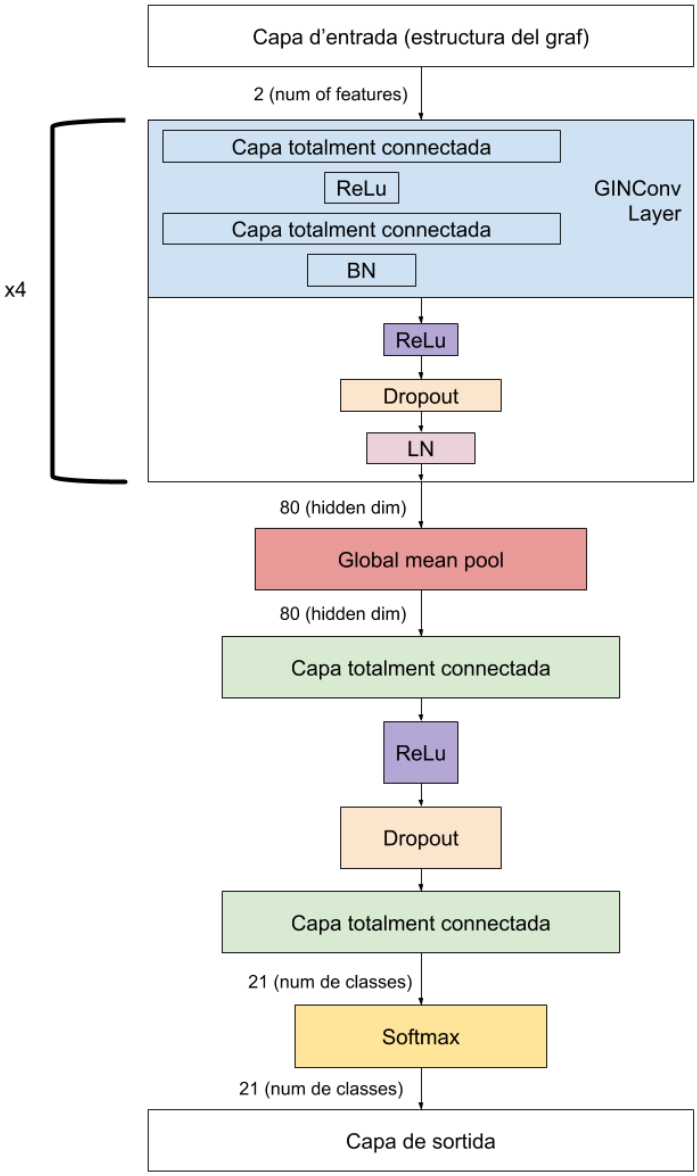

Figura 8.1: Arquitectura del model GIN implementat

8.2 Descripció de l'arquitectura del model implementat

La capa d'entrada consisteix en els grafs. La mida de l'entrada és el nombre de característiques (features) dels grafs d'entrada. Per tant, la mida serà 2, ja que només es tenen dues característiques en el graf (a part de la seva estructura en si), les quals són la

puntuació de cada vèrtex (comentari) i el valor del sentiment que té un vèrtex v_j sobre el vèrtex v_i al que respon.

Llavors, la mida de les capes ocultes (hidden dim) s'ha definit en l'experimentació. La mida que ha donat millors resultats ha sigut 80, tal com apareix en el diagrama.

Finalment, la capa de sortida té com a mida el nombre de classes possibles en les quals es poden classificar els grafs, el qual és 21.

A continuació s'explica el dignificat de cada ítem de l'arquitectura:

- *ReLu* és la funció d'activació ReLu.
- *BN* significa *Batch Normalization*
- *LN* significa *Layer Normalization*
- La *capa totalment connectada* concretament aplica una transformació lineal a les dades entrants. El nombre de neurones de la capa es defineix en l'experimentació.
- *Global mean pool* és la capa on es duu a terme l'operació de mean pooling.
- *Softmax* és la funció softmax

Capítol 9

Experimentació

9.1 Característiques del sistema

Les característiques del sistema en el que s'ha dut a terme l'experimentació són les següents:

Intel® Core™ i5-6400 CPU @ 2.70GHz x 4 GeForce GTX 1650 SUPER: 1280 CUDA cores i 4 GB de memòria 7,7 GB usables de memòria RAM

Ubuntu 20.04.1 LTS 64-bit

9.2 Hiperparàmetres escollits

El dataset de 6395 exemples i $\alpha = 0.5$ és el que conté més dades, una representació prou fidel de les converses originals (sense filtrar), i té el percentatge d'encerts més alt (en comparació amb valors α menors). Per tant aquest és el dataset escollit, i sobre aquesta base s'ha optimitzat el model de xarxa probant múltiples valors per cada hiperparàmetre. La combinació d'hiperparàmetres que han aconseguit la precisió més alta és la següent:

- *Mida del batch*: 1474
- *Learning rate*: 0.1
- S'ha utilitzat el *Scheduler*, el qual s'ha configurat per tal que quan la precisió arribi a 0.235, el *learning rate* disminueixi amb un factor de 0.5 cada vegada que s'augmenti el valor de la precisió en 0.005.
- *Nombre d'epochs*: 400
- *Nombre d'unitats per capa oculta*: 80
- *Nombre de capes ocultes*: 4

- *Dropout*: 0.2

Si no s'especifiquen els hiperparàmetres utilitzats en qualsevol apartat de l'experimentació, aquests seran els valors dels hiperparàmetres en el model de xarxa neuronal.

9.3 Elecció i precisions dels datasets

S'han elegit conversacions antigues (anys 2018 i 2019), ja que ens interessa analitzar converses on tothom qui vulgui hagi pogut contestar, i que tots els paràmetres necessaris siguin visibles.

Al cap de 6 mesos reddit arxiva les converses i ja no deixa que ningú més respongui ni valori els comentaris. Llavors també hi ha una opció amb la qual els moderadors poden amagar els scores dels comentaris durant un màxim de 24 hores. Per tant sempre s'ha descarregat converses de més de 6 mesos però no molt antigues.

El dataset de 6395 exemples conté converses de cada dia del anys 2018 i 2019. Hi ha un mínim de 3 i un màxim de 10 converses per cada dia. Totes les converses tenen menys de 1300 comentaris (majoritàriament menys de 1000 comentaris). La mitjana de converses per dia és de 8.76.

Tant el dataset de 3440 exemples com el de 2527 exemples contenen converses de cada dia de l'any 2019, tot i tenir més o menys converses per dia.

9.3.1 Distribucions dels datasets

En les imatges 9.1, 9.2 i 9.3 es poden veure les distribucions del dataset de 6395, 3440 i 2527 exemples, respectivament.

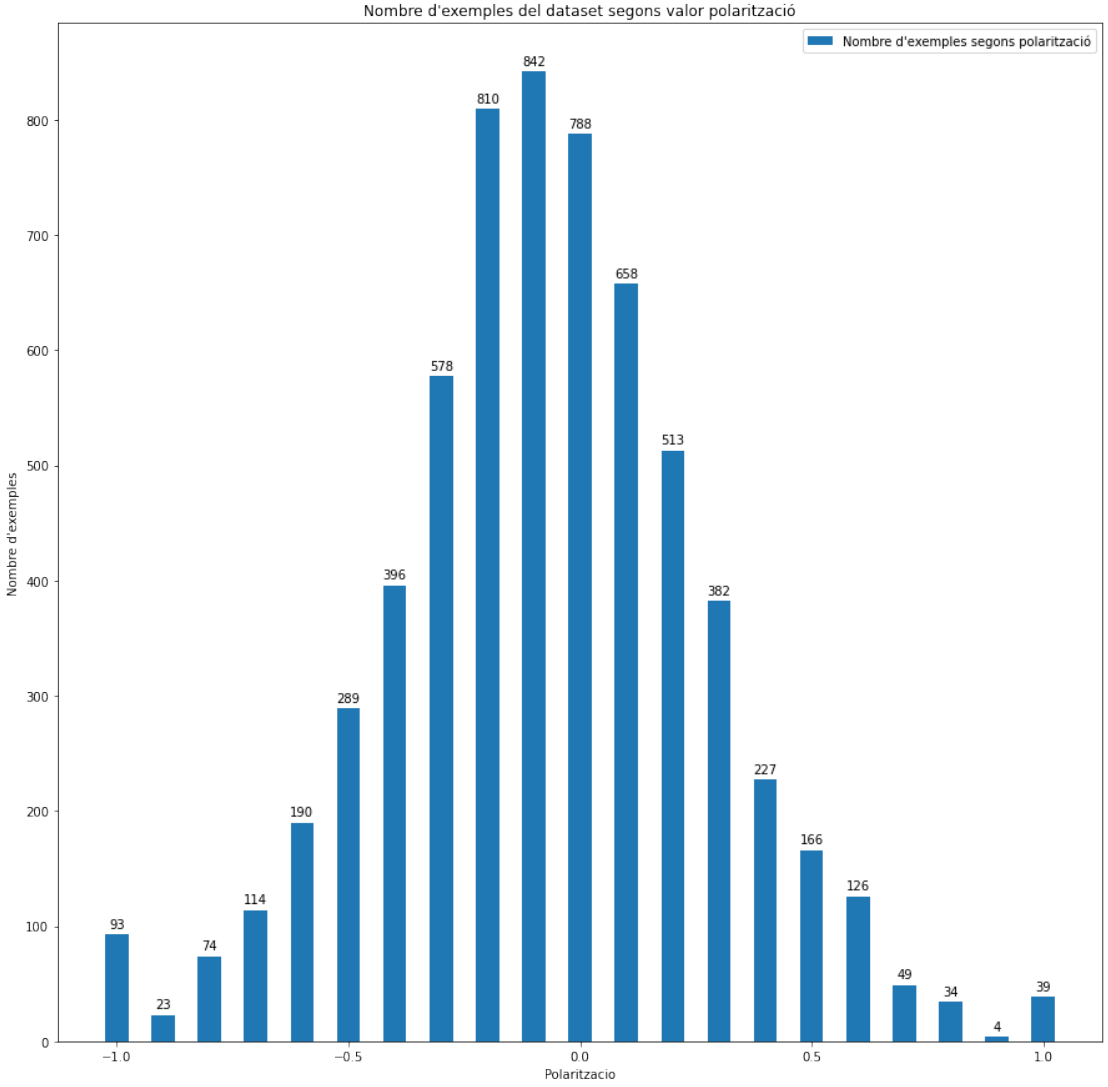

Figura 9.1: Distribució dels exemples al dataset de 6395 exemples, amb $\alpha = 0.5$

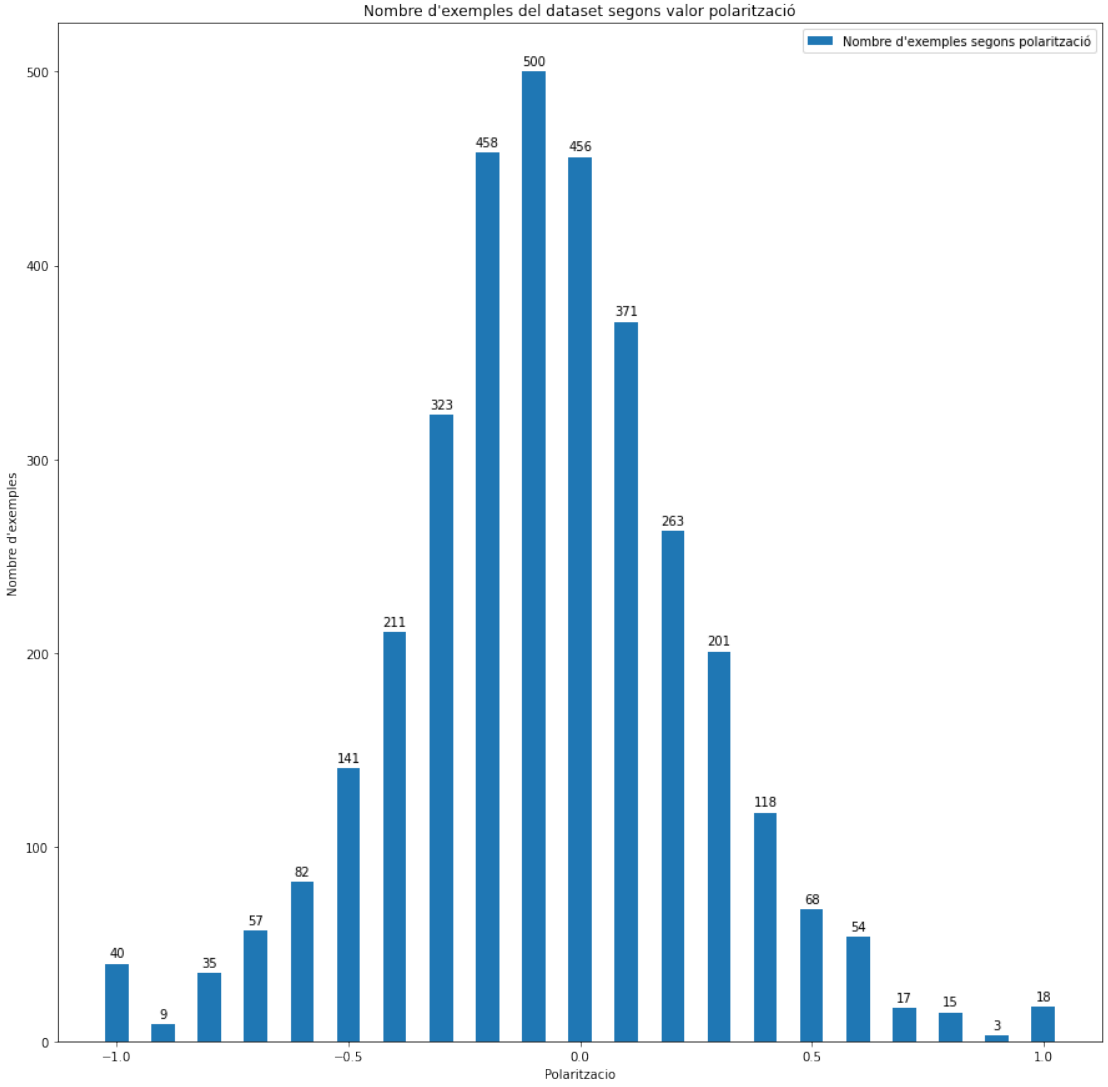

Figura 9.2: Distribució dels exemples al dataset de 3440 exemples, amb $\alpha = 0.5$

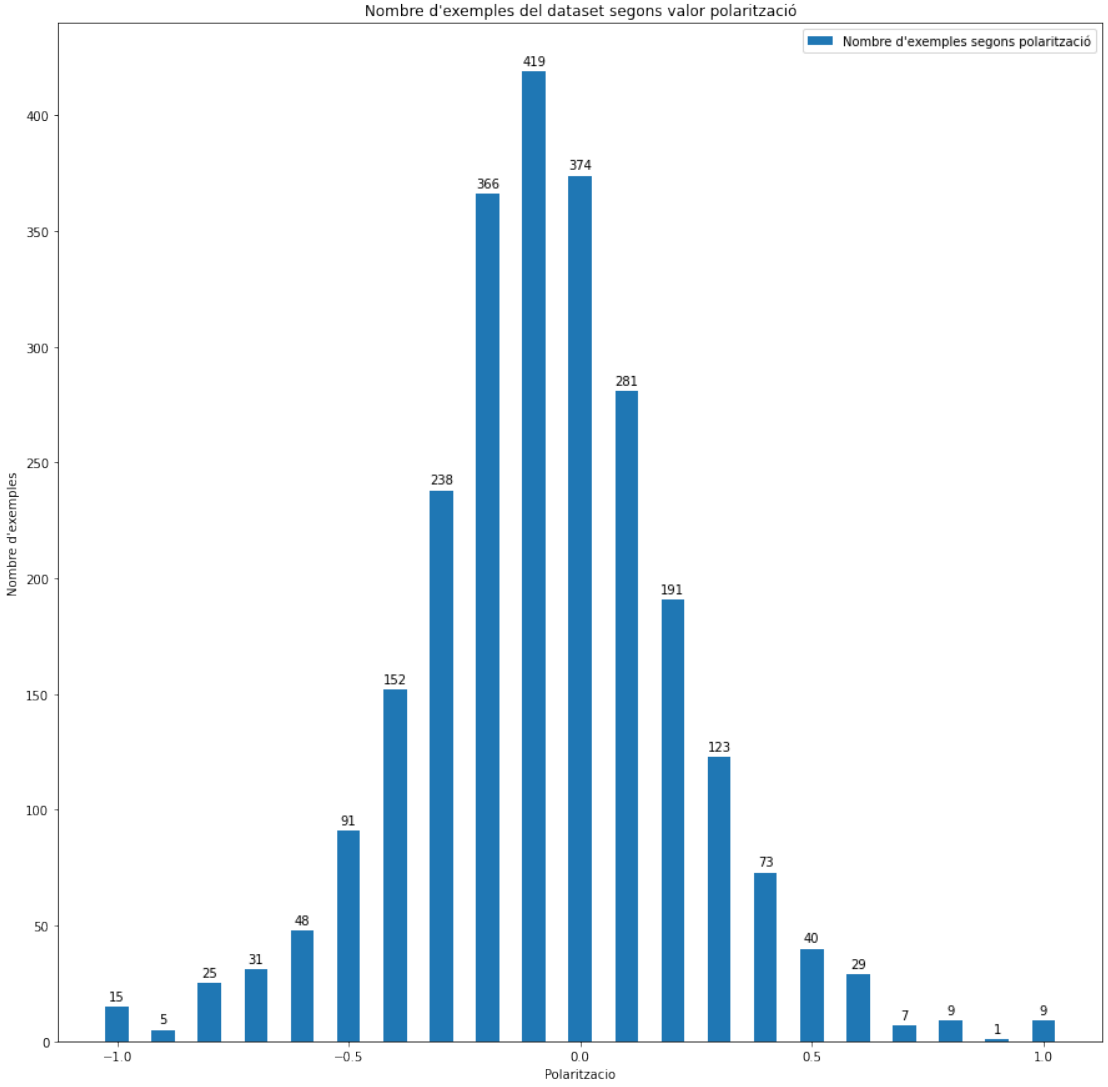

Figura 9.3: Distribució dels exemples al dataset de 2527 exemples, amb $\alpha = 0.5$

9.3.2 Desequilibri dels datasets segons l'ordre mig

Cal destacar que hi ha un desequilibri en el nombre d'exemples de cada classe, el qual s'ha de tenir en compte per tal que el model tingui en compte les classes amb menys exemples. Com més comentaris (vèrtexs) tenen les converses, la polarització tendeix a la neutralitat (0), i les polaritzacions més extremes (-1, +1), només ocorren en conversacions amb molt pocs comentaris.

Valor Polarització	ID Classe	Ordre mig
-1.0	10	7.935
-0.9	9	23.565
-0.8	8	17.621
-0.7	7	21.245
-0.6	6	22.084
-0.5	5	27.429
-0.4	4	40.033
-0.3	3	48.993
-0.2	2	57.637
-0.1	1	61.549
0.0	0	55.992
0.1	11	46.879
0.2	12	37.563
0.3	13	24.79
0.4	14	26.044
0.5	15	22.723
0.6	16	17.317
0.7	17	18.836
0.8	18	16.764
0.9	19	18.0
1.0	20	7.307

Taula 9.1: Ordre mig dels grafs segons la polarització. S'ha utilitzat el dataset de 6395 exemples amb $\alpha = 0.5$

Això es pot veure en la taula 9.1, on es mostra la mitjana de l'ordre dels grafs en cada una de les 21 classes per al dataset de 6395 exemples amb $\alpha = 0.5$.

En la taula 9.2, on es mostra la mitjana de l'ordre dels grafs en cada una de les 21 classes per al dataset de 6395 exemples amb $\alpha = 0.5$, també s'observa el mateix comportament que en la taula 9.1, tot i què amb valors més grans, lògic ja que les converses estan menys filtrades (contenen una part més gran dels comentaris de la conversa original).

En canvi, en la taula 9.3, on es mostra la mitjana de l'ordre dels grafs en cada una de les 21 classes per al dataset de 6395 exemples amb $\alpha = 1$, no té el mateix comportament que quan s'ha utilitzat un valor de α menor.

Valor Polarització	ID Classe	Ordre mig
-1.0	10	11.92
-0.9	9	24.666
-0.8	8	24.419
-0.7	7	28.672
-0.6	6	33.165
-0.5	5	38.128
-0.4	4	41.211
-0.3	3	68.379
-0.2	2	90.161
-0.1	1	118.151
0.0	0	114.266
0.1	11	84.479
0.2	12	62.410
0.3	13	44.740
0.4	14	40.123
0.5	15	32.707
0.6	16	31.483
0.7	17	29.266
0.8	18	25.363
0.9	19	31.25
1.0	20	8.666

Taula 9.2: Ordre mig dels grafs segons la polarització. S'ha utilitzat el dataset de 6395 exemples amb $\alpha = 0.1$

Valor Polarització	ID Classe	Ordre mig
-1.0	10	4.47
-0.9	9	18.666
-0.8	8	11.037
-0.7	7	9.928
-0.6	6	17.861
-0.5	5	16.777
-0.4	4	19.485
-0.3	3	14.929
-0.2	2	13.463
-0.1	1	10.933
0.0	0	8.487
0.1	11	11.111
0.2	12	6.213
0.3	13	6.172
0.4	14	8.546
0.5	15	4.739
0.6	16	6.441
0.7	17	5.696
0.8	18	7.777
0.9	19	12
1.0	20	3.77

Taula 9.3: Ordre mig dels grafs segons la polarització. S'ha utilitzat el dataset de 6395 exemples amb $\alpha = 1$

9.3.3 Distribucions del dataset de 6395 exemples amb diferents α

Aquesta és la segona prova on s'intenten veure les diferències en les distribucions dels exemples d'un dataset segons el paràmetre α. En la figura 9.4 es pot apreciar com la distribució dels exemples canvia lleugerament respecte a la versió de $\alpha = 0.5$. En canvi, en la figura 9.4 la distribució canvia notablement, augmentant amb força els casos extrems. Cal tenir en compte que el nombre real d'exemples en aquest cas ha baixat de 6395 a 6011, ja que establint el filtre a $\alpha = 1$ hi ha hagut exemples que s'han reduit al comentari arrel, i aquests casos no es tenen en compte.

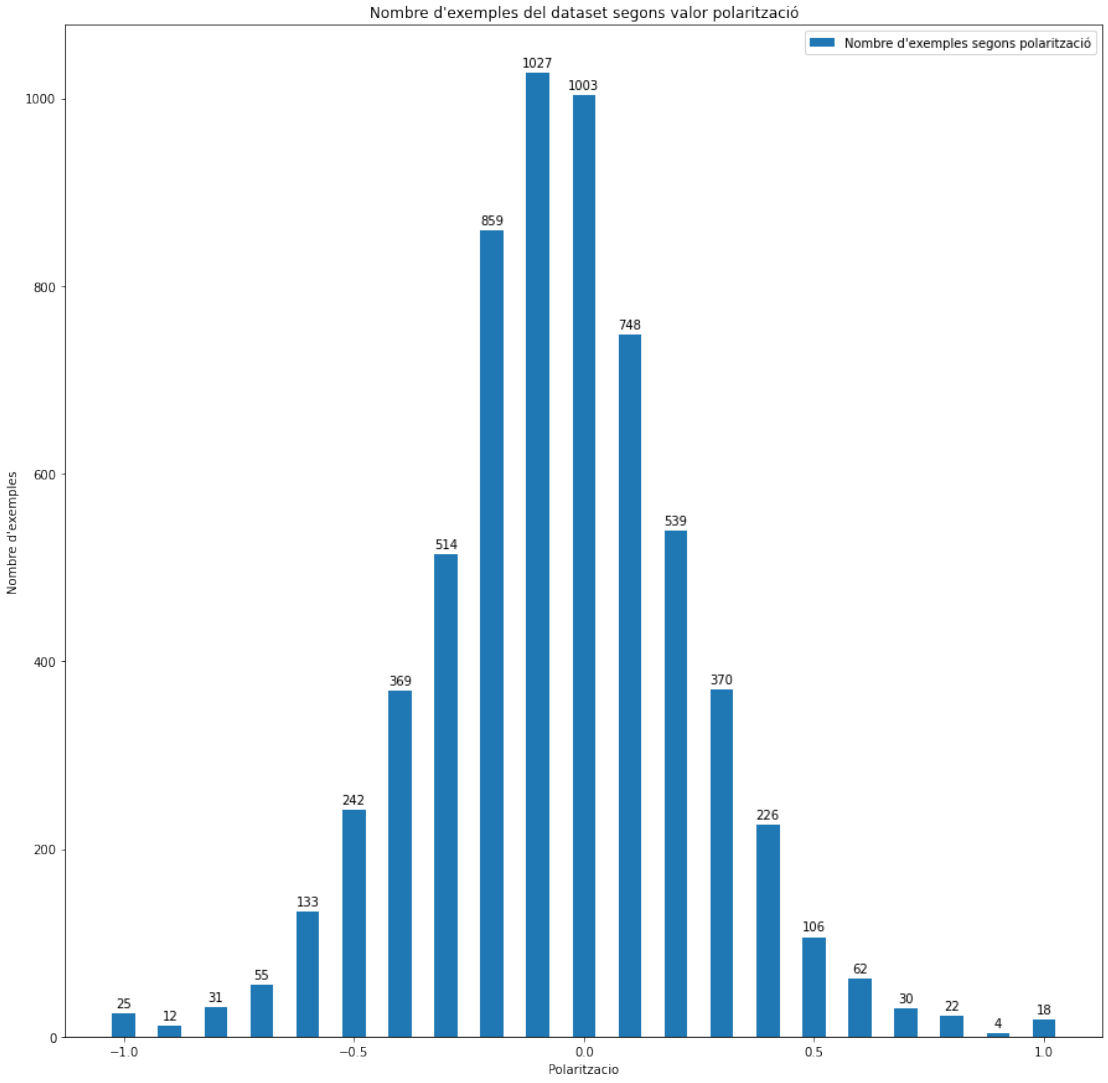

Figura 9.4: Distribució dels exemples al dataset de 6395 exemples, amb $\alpha = 0.1$

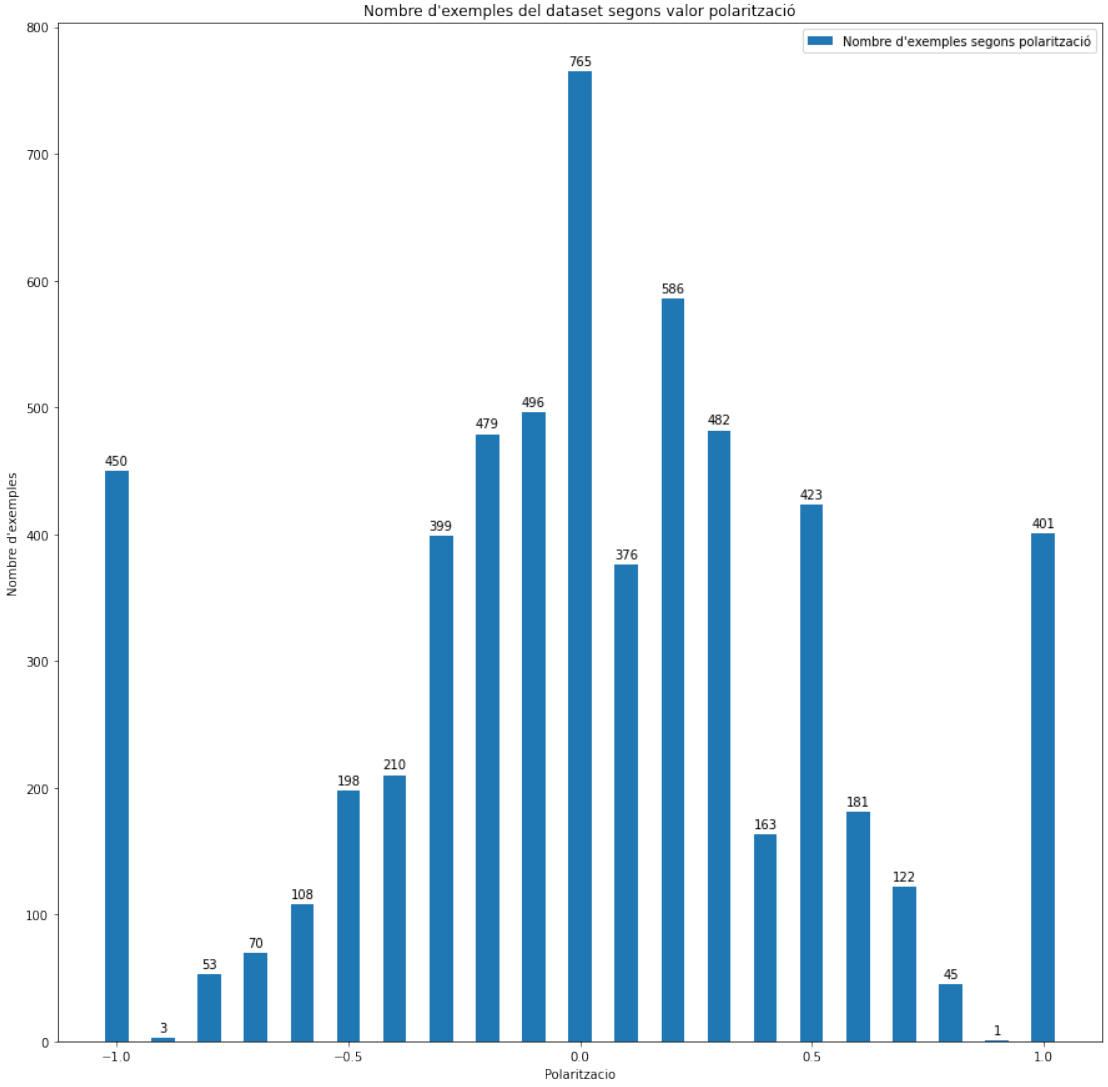

Figura 9.5: Distribució dels exemples al dataset de 6395 exemples, amb $\alpha = 1$

Exemples del dataset	Mitjana de precisions	Mitjana de precisions màximes	Precisió màxima
2527	0.198	0.243	0.281
3440	0.185	0.229	0.253
6395	0.198	0.236	0.256

Taula 9.4: Precisió del model en diferents datasets, utilitzant stratified 10-fold-cross-validation

Valor α	Mitjana de precisions	Mitjana de precisions màximes	Precisió màxima
0.1	0.18	0.209	0.228
0.5	0.198	0.236	0.256
1	0.403	0.506	0.543

Taula 9.5: Precisió del model amb diferents α, utilitzant stratified 10-fold-cross-validation

9.3.4 Precisions dels datasets amb $\alpha = 0.5$

En la taula 9.4 es mostra la precisió del model utilitzant 3 datasets. Resultats utilitzant hiperparàmetres ja optimitzats.

Havent realitzat un gran nombre de proves, s'aconsegueix un augment en la precisió del model utilitzant el dataset de 6395 respecte al de 3440, però no sobre el de 2527 exemples. Com és molt probable que els millors resultats del dataset de 2527 exemples siguin perquè el nombre d'exemples en les classes menys comunes és molt baix, s'escull el dataset de 6395 com a dataset definitiu, ja que és una representació més completa de les dades.

9.3.5 Precisions del dataset de 6395 exemples amb diversos α

En la taula 9.5 es mostra la precisió del model utilitzant el dataset de 6395 exemples amb diferents valors de α.

Com es pot observar, com més augmenta el valor de α, més alta ha sigut la precisió del model. Com en els apartats anteriors s'ha vist que el dataset amb valor $\alpha = 1$ obté unes polaritzacions notablement diferents a valors menors de α, i tenint un comportament també diferenciat, no es pot considerar el dataset amb $\alpha = 1$ com un dataset que representi el model de dades de forma fiable, per tant es descartaria encara que la precisió sigui molt més alta. La precisió és més alta, ja que els grafs són d'una mida més petita i per tant estructures reduïdes i amb més similituds.

Algorisme	Precisió	Precisió màxima	Precisió màxima amb marge +-1
Random guess	0.0872	-	-
Zero rule	0.1316	-	-
Model implementat	-	0.2645	0.526

Taula 9.6: Comparació de la recisió dels classificadors genèrics amb la del model GIN implementat

9.4 Precisió del model respecte altres classificadors

S'utilitza stratified-10-fold-cross-validation i el testset és un 10% del dataset complet.

Com no tenim una *baseline* (és a dir, altres papers amb models que provin el mateix dataset), podem utilitzar 2 classificador genèrics per tenir un punt de comparació, determinant realment si el model implementat està aprenent sobre les dades, i quina és la qualitat d'aquest.

1) **Random guess**:

$$accuracy = \frac{\sum_{i=1}^{c} p_i n_i}{\sum n_i}$$

on
n_i és el nombre d'exemples en el dataset de la classe i
$p_i = P(\text{estar en la classe } i) = \frac{n_i}{\sum_{x=1}^{c} n_x}$

Per tant, la precisió del classificador Random Guess serà:

$$accuracy = \frac{557.6686473807662}{6395} = 0.0872$$

2) **Zero rule**:

$$accuracy = \frac{max_{1 \leq i \leq c} n_i}{\sum n_i}$$

La precisió del classificador Zero Rule és:

$$accuracy = \frac{842}{6395} = 0.1316$$

En la taula 9.6 es pot observar la comparació dels classificadors.

Capítol 10

Conclusions

S'ha aconseguit l'objectiu de crear un model de xarxa neuronal que accepti qualsevol tipus de graf com a entrada i predigui la polarització. Tot i això, la accuracy de la predicció és més baixa de l'esperada (26.45% amb 1 decimal de predicció (21 possibles opcions) i 50% amb un marge de 0.1). S'ha pogut comprovar que el model aconsegueix aprendre sobre les dades, però és una precisió molt baixa per poder arribar a utilizar aquest model en la pràctica, ja que normalment es volen aconseguir precisions superiors al 90% de precisió.

Aquests resultats venen després d'utilitzar un dels models que millors resultats ha obtingut en altres problemes de classificació de grafs, i s'ha fet un esforç per treure el millor rendiment del model possible mitjançant l' hiperparameter tuning.

Considero que aquests resultats són deguts a l'alta complexitat del model de dades. Els exemples existents de GNNs de classificació que arriben a precisions del 80%oel90% contenen models de dades més senzills i menys de 21 classes per classificar cada vèrtex.

La gran majoria del temps dedicat en el treball ha sigut en la part teòrica, ja que prèviament només coneixia teoria bàsica sobre xarxes neuronals, i ha calgut aprofundir força en aquest tema, sobretot en l'apartat de les xarxes neuronals per a grafs. A més a més, com la informació sobre GNNs és molt recent i la informació és escassa, la informació recopilada s'ha basat fonamentalment en papers. També s'ha après en aquest treball de manera pràctica a desenvolupar, optimitzar i entrenar una xarxa neuronal, ja que només es tenia coneixements teòrics sobre aquesta matèria.

10.1 Possibles millores

Un dels possibles canvis que es podrien dur a terme serien:

- Aconseguir un hyperparameter-tuning millor. - Un model de xarxa neuronal millor. - Afegir més exemples al dataset. El model de xarxa neuronal s'ha definit per tal d'

adquirir un al nivell de complexitat. Com que s'han vist millores notables a l'afegir més dades i el model de xarxa no sembla ser encara un limitador, es possible que augmentar la mida del dataset millori la precisió del model, sobretot en el cas de les polaritzacions menys comunes, ja que - Canviar el model de dades, fer-lo més senzill i que siguin més evidents les característiques del graf que comporten una polarització o una altra.

En els 4 casos una millora no està assegurada, caldria provar-ho.

A més a més, com que les GNNs són un camp en constant evolució, és molt possible que en un futur pròxim sorgeixin nous models i mètodes d'optimització que aconsegueixin millorar la precisió. Llavors utilitzant la mateixa base podríem veure millores en la precisió.

També: - Focal loss. Feature que compareixen 2 classes, va a la que tingui + exemples. - Crear converses artificials a partir de les originals treient nodes de conversa original per forçar una polarització determinada: per intentar compensar la gran diferència en el nombre d'exemples en algunes polaritzacions en concret, per exemple -0.1 (molts exemples, 841 exemples d'un total de 6395) i 0.9 (molts pocs exemples, només 4 exemples d'un total de 6395). Òbviament el millor seria compensar la diferència buscant conversacions de Reddit originals que tinguin les polaritzacions menys comunes i treure'n algunes de les polaritzacions més comunes, però alguns tipus de polaritzacions són extremadament escasses (com la 0.9).

10.2 Possibles ampliacions en el treball

L'ampliació més útil d'aquest treball seria provar el model utilitzant grafs amb cicles, per exemple, resoldre el mateix problema però utilitzant el model de dades centrat en els autors. Com s'ha explicat en el treball, no hi ha solució polinòmica en aquest cas i una xarxa neuronal prèviament entrenada pot classificar les dades en temps polinòmic.

Bibliografia

[1] T. ALSINET, J. ARGELICH, R. BÉJAR, and S. MARTÍNEZ, "An argumentation approach for agreement analysis in reddit debates," 2018.

[2] T. J. M. Bench-Capon, "Value-based argumentation frameworks," pp. 443–454, 2002. [Online]. Available: https://arxiv.org/abs/cs/0207059

[3] P. M. P. M. Dung and F. Toni, "Computing ideal sceptical argumentation," pp. 642–674, 2007. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S000437020700080X

[4] T. ALSINET, J. ARGELICH, R. BÉJAR, C. FERNÀNDEZ, C. MATEU, and J. PLANES, "Weighted argumentation for analysis of discussions in twitter," 2017. [Online]. Available: https://repositori.udl.cat/handle/10459.1/59372

[5] T. ALSINET, J. ARGELICH, R. BÉJAR, and J. CEMELI, "A distributed argumentation algorithm for mining consistent opinions in weighted twitter discussions," 2019. [Online]. Available: https://repositori.udl.cat/handle/10459.1/67673

[6] M. F. Jin-Yi Cai and N. Immerman, "An optimal lower bound on the number of variables for graph identification," 1992. [Online]. Available: https://people.cs.umass.edu/~immerman/pub/opt.pdf

[7] G. J. W. Martin Grohe, Gaurav Rattan, "Graph similarity and approximate isomorphism," 2018. [Online]. Available: https://arxiv.org/abs/1802.08509

[8] A. C. T. M. H. G. M. Franco Scarselli, Marco Gori, "The graph neural network model," 2009. [Online]. Available: https://persagen.com/files/misc/scarselli2009graph.pdf

[9] T. L. Y. B. X. B. Vijay Prakash Dwivedi, Chaitanya K. Joshi, "Benchmarking graph neural networks," 2020. [Online]. Available: https://arxiv.org/abs/2003.00982

[10] J. L. S. J. Keyulu Xu, Weihua Hu, "How powerful are graph neural networks?" 2018. [Online]. Available: https://arxiv.org/abs/1810.00826